VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH LABORATORNÍCH ÚLOH PRO VÝUKU SÍŤOVÝCH TECHNOLOGIÍ A PROTOKOLŮ
LABORATORY EXERCISES EXPLAINING NETWORK TECHNOLOGIES AND PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Coufal**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Lukáš Langhammer, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Tomáš Coufal ***ID:*** 164250
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Návrh laboratorních úloh pro výuku síťových technologií a protokolů

POKYNY PRO VYPRACOVÁNÍ:

Zadání práce spočívá v návrhu tří nových laboratorních úloh včetně kompletních návodů vhodných pro studenty zahrnující výchozí scénář, doplňující úkoly a kontrolní otázky. Nastudujte problematiku komunikačních protokolů a vyberte vhodné simulační prostředí, ve kterém budou laboratorní úlohy vytvořeny. Obsah úloh zaměřte na okruhy: rozbor aplikačních protokolů (FTP, HTTP, SMTP, DNS, popřípadě další), srovnání transportních protokolů TCP, UDP, SCTP, demonstrace funkce kvality služeb (QoS), porovnání technologií Ethernet a WLAN, ATM, Frame Relay atd. Časová náročnost každé úlohy musí být přibližně dvě hodiny.

DOPORUČENÁ LITERATURA:

[1] FOROUZAN, B. A. TCP/IP Protocol Suite. Fourth edition, Boston: McGraw-Hill Higher Education, 2010, 979 stran. ISBN 978-0-07-337604-2.

[2] JEŘÁBEK, J. Komunikační technologie. Skriptum FEKT Vysoké učení technické v Brně, 2016. s. 1-172.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. Lukáš Langhammer, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Diplomová práce se zabývá tvorbou laboratorních úloh v prostředí ns-3. Každá ze tří úloh se skládá z teoretického úvodu a návodu k vytvoření simulace. Tématem první úlohy je směrovací protokol BGP. Druhá úloha je zaměřena na transportní protokoly TCP, UDP, SCTP. V poslední úloze jsou simulovány síťové prvky a základní topologie. Simulován je také protokol ARP a RIPv2.

Klíčová slova

BGP, TCP, UDP, SCTP, ARP, RIPv2, topologie, směrovač, přepínač, ns-3, DCE, Quagga, Wireshark.

Abstract

Diploma thesis deals with creation of laboratory exercises in ns-3 environment. Each one of three exercises consists of theoretical introduction and instructions to carry out the simulation. The first exercise´s topic is routing protocol BGP. The second exercise is focused on transport protocols TCP, UDP, SCTP. In the last exercise, the basic network devices and topologies are simulated. The ARP and RIPv2 protocols are simulated as well.

Keywords

BGP, TCP, UDP, SCTP, ARP, RIPv2, topologies, router, switch, ns-3, DCE, Quagga, Wireshark.

Bibliografická citace:

COUFAL, T. Návrh laboratorních úloh pro výuku síťových technologií a protokolů. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2019. 114 s. Vedoucí diplomové práce Ing. Lukáš Langhammer, Ph.D..

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma návrh laboratorních úloh pro výuku síťových technologií a protokolů, jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **16. května 2019** ……

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Lukáši Langhammerovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **16. května 2019** ……

podpis autora

Obsah

Seznam symbolů a zkratek

Zkratky:

ARP	Address Resolution Protocol
AS	Autonomous system
BGP	Border Gateway Protocol
DCE	Direct Code Execution
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
FTP	File Transfer Protocol
GNU GPL	General Public License
HTTP	Hypertext Transfer Protocol
IGP	Interior Gateway Protocol
IGMP	Internet Group Management Protokol
NLRI	Network Layer Reachability Information
Ns-2	Network simulator 2
Ns-3	Network simulator 3
OSPF	Open Shortest Path First
RFC	Request For Comments
RIB	Routing Information Base
RIP	Routing Information Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SCTP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Seznam obrázků

Seznam tabulek

1. ÚVOD

V diplomové práci je cílem vytvořit tři laboratorní úlohy pro výuku síťových technologií a protokolů. V první části práce je představen simulátor ns-3, DCE a balík Quagga. Popsána je i instalace simulátoru na linuxových platformách. Samotné laboratorní úlohy se skládají z teoretického úvodu a praktického návodu k vytvoření simulace v ns-3.

První laboratorní úloha se zabývá směrovacím protokolem BGP, který je důležitou součástí dnešního internetu, ačkoli se k němu běžný uživatel vůbec nedostane. Pro návrh laboratorní úlohy je použit simulátor ns-3 s podporou přímého vykonávání kódu DCE a balíkem Quagga, který rozšiřuje podporu směrovacích protokolů v ns-3. V teoretickém úvodu jsou představeny základní vlastnosti protokolu BGP, jsou popsány zprávy a směrovací proces. V laboratorní úloha je zaměřena na konfiguraci sousedů, zasílané zprávy je možné zobrazit v programu Wireshark. Vytvořen je také graf zatížení linek protokolem BGP.

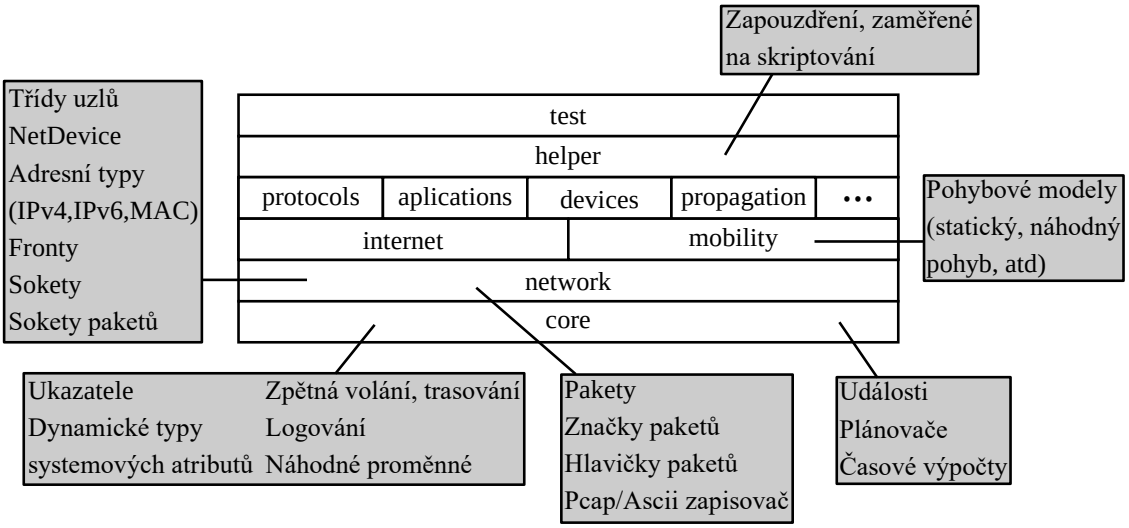
Druhá laboratorní úloha se zabývá transportními protokoly. V teoretické části jsou rozebrány vlastnosti a funkce protokolů UDP, TCP a SCTP. Pro všechny protokoly jsou také rozebrány informace, které jsou přenášeny v hlavičkách. První část laboratorní úlohy je zaměřena na srovnání protokolů TCP a UDP. Protokoly jsou provozovány na vlastní síti a jsou sledovány jejich reakce na chybovost a výpadek. Ve společné síti je poté simulováno přetížení v síti. Závěrem úlohy je simulace protokolu SCTP pomocí DCE v pokročilém módu. Je zde předvedeno navázání a ukončení spojení a komunikace mezi dvěma uzly. Ukázány jsou také některé bloky, které jsou v SCTP paketu přenášeny.

Poslední úloha se zabývá srovnáním mezi směrovačem a přepínačem. V rámci tohoto srovnání je představen protokol ARP a ukázána funkce směrování. Simulovány jsou také jednoduché topologie jako hvězda, strom a obecná topologie. V poslední části úlohy je ve stromové topologii povoleno směrování pomocí směrovacího protokolu RIPv2, který je v ns-3 podporován včetně signalizace. Je předvedena funkce směrování, reakce na výpadky a použitá metrika. Ukázány jsou také směrovací tabulky.

2. NS-3

Ns-3 je síťový simulátor diskrétních událostí, určený především pro výzkum a vzdělávání. Je dostupný pro volné použití pod licencí GNU GPLv2 [1]. Vývoj simulátoru začal v roce 2006 jako náhrada za simulátor ns-2. I přes podobnost v názvech jde ale kompletně o nový simulátor, který není zpětně kompatibilní s ns-2 [2]. Vývoj simulátoru stále probíhá a neustále se pracuje na zlepšení podpory. V současné době je dostupný ve verzi 3.29.

Ns-3 je v podstatě knihovna propojená s hlavním programem, který definuje simulovanou topologii a spouští samotný simulátor. Hlavní program může být psaný v jazyce C++ případně Phyton [3]. Strukturu simulátoru zobrazuje Obr. 2-1.



Obr. 2-1: Struktura ns-3 [3]

2.1 Základní objekty

Před používáním ns-3 je vhodné se seznámit se základními objekty, které ns-3 poskytuje. Ty zavádí určitou míru abstrakce systému a slouží ke snadnější konfiguraci a také značnou měrou zpřehledňují výsledný kód simulace [4].

2.1.1 Uzel

Základní abstrakce výpočetního zařízení v ns-3 se nazývá uzel. V jazyce C++ je reprezentována třídou Node a poskytuje metody pro správu a reprezentaci výpočetních

zařízení v simulacích. Uzel je v podstatě počítač, ke kterému jsou přidávány funkce (aplikace, protokolové zásobníky, …) [4].

2.1.2 Kanál

Abstrakce komunikačního kanálu v ns-3 zajišťuje C++ třída Channel. Třída poskytuje metody pro správu a připojování uzlů k síti. V reálném světě to můžeme připodobnit připojením počítače do lokální sítě prostřednictvím technologie Ethernet. V ns-3 je k dispozici několik specializovaných kanálů. Mezi ty patří CsmaChannel, PointToPointChannel a WifiChannel [4].

2.1.3 Aplikace

Program, který generuje aktivitu, je v ns-3 reprezentován třídou Apliccation. Tato třída zajišťuje požadovanou funkčnost a metody k tomu potřebné. Pro předpoklad můžeme uvést jednoduchou UDP aplikaci která pracuje v režimu klient/server názvem UdpEchoClientApplication a UdpEchoServerApplication [4].

2.1.4 Síťové rozhraní

V reálném prostředí je pro připojení počítače do sítě nutná síťová karta. V prostředí ns-3 se abstrakce síťového zařízení vztahuje jak na softwarový ovladač, tak na simulovaný hardware. Abstrakce je zajištěna C++ třídou NetDevice a do uzlu je nutná jeho instalace ke komunikaci skrze komunikační kanál. Stejně jako v reálném počítači může být uzel připojen k několika komunikačním kanálům prostřednictvím několika zařízení NetDevice. Opět je zde několik specializovaných rozhraní (CsmaNetDevice, PointToPointNetDevice a WifiNetDevice), které korespondují s kanály [4].

2.1.5 Topology helpers

Pro usnadnění tvorby sítě slouží v ns-3 topology helpers. Ulehčuje připojování síťového rozhraní k uzlu a následně ke kanálu automatickým přiřazením adres (IP, MAC,…) [4]. Bez použití topology helperu by k těmto operacím bylo potřeba mnoho příkazu, helper tedy ulehčuje práci s vytvořením modelu sítě a také zpřehledňuje kód.

2.2 Přímé vykonávání kódu DCE

DCE (Direct Code Execution) je modul, který umožňuje ns-3 využívat stávajících implementací systému. Například síťových protokolů či aplikací bez změn zdrojového kódu. DCE je možné používat ve dvou režimech. Základní mód používá sadu TCP/IP implementovanou v systému ns-3. Pokročilý mód potom sadu síťových protokolů implementovanou v operačním systému [5]. Pro zjednodušení konfigurace bude v laboratorní úloze BGP použit základní mód. Pro simulaci protokolu SCTP, je nutné použít pokročilý mód. K propojení simulátoru slouží třídy DceManagerHelper a DceApplicationHelper, které zajišťují funkce pro správný běh simulace.

Pro rozšíření DCE je možné využít i dalších přídavných balíků. Mezi takové patří i balík Quagga který zajišťuje podporu několika směrovacích protokolů [5].

2.3 Quagga

Balík Quagga slouží k zajištění směrování pro operační systémy Linux, Solaris, NetBSD. Quagga je založen na projektu GNU Zebra a jeho architektura se skládá z démonů, zajišťujících směrování [6].

Funkce balíku Quagga v ns-3 jsou v dnešní době stále ve vývoji. Stále není podporováno množství funkcí směrovacích protokolů [7]. Výčet protokolů, které v ns-3 omezeně fungují, je uveden v Tab. 2-1.

Tab. 2-1 : Protokoly podporované balíkem Quagga pro ns-3 [7]

	Základní mód (ns-3)	Pokročilý mód (ns-3-linux)
Rtadvd (zebra)	-	OK
RIPv1/v2 (ripd)	-	OK
RIPng (ripngd)	-	OK
OSPFv2 (ospfd)	OK	OK
OSPFv3 (ospf6d)	-	OK
BGP (bgpd)	OK	OK
BGP+ (bgpd)	-	OK

Spojení quaggy se simulátorem zajišťuje třída QuaggaHelper, která umožňuje nastavení směrování pomocí kódu vytvořeného v ns-3.

2.4 Instalace DCE Quagga

Tato kapitola popisuje kompletní instalaci ns-3 DCE s balíkem Quagga. Před samotnou instalací je třeba systém připravit instalací několika balíčků. V systému Ubuntu k tomu slouží příkaz **apt-get install**. Balíčky, které je třeba stáhnout a nainstalovat, jsou následující: ***autoconf, automake, flex, git-core, wget, g++, libc-dbg, bison, indent, pkgconfig, libssl-dev, libsysfs-dev, gawk*** [7]**.**

Nyní je již možné přejít k samotné instalaci ns-3 DCE Quagga. K instalaci pomocí Mercurial je nejdříve nutné stáhnout konfigurační soubory do adresáře bake a definovat proměnné následujícími příkazy:

```
hg clone http://code.nsnam.org/bake bake
export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```

Po vykonání těchto kroků je třeba vytvořit adresář do kterého se následně stáhne a sestaví program.

```
mkdir dce
cd dce
bake.py configure -e dce-ns3 -e dce-quagga
bake.py download
bake.py build
```

Pro použití pokročilého módu ns-3 a použití nativních protokolů linux je třeba příkaz bake.py configure -e dce-ns3 -e dce-quagga nahradit příkazem bake.py configure -e dce-linux- -e dce-quagga [7]. Po úspěšném stažení a sestavení je možné instalaci zkontrolovat příkazem:

```
cd source/ns-3-dce
./test.py -s dce-quagga
```

V případě správné funkčnosti by měl výstup těchto příkazů vypadat zhruba takto [7]:
PASS: TestSuite dce-quagga 9.775 s
1 of 1 tests passed (1 passed, 0 skipped, 0 failed, 0 crashed, 0 valgrind errors).
Instalace ns-3 DCE s podporou balíku Quagga je tedy připravena k použití.

2.5 Instalace DCE v pokročilém módu

Pro simulaci protokolu SCTP je třeba využít DCE v pokročilém módu. Instalace je velmi podobná jako v předchozí kapitole. Nejdříve je nutné stáhnout konfigurační soubory do adresáře bake a definovat proměnné [8]:

```
hg clone http://code.nsnam.org/bake bake
export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```

Dále se vytvoří adresář a do něj se nainstaluje DCE v pokročilém módu [8]:

```
mkdir dce
cd dce
bake.py configure -e dce-linux-|verze|
bake.py download
bake.py build
```

Dále je třeba použít upravené jádro systému Linux. V našem případě se jedná o *Libos.* Následujícími příkazy získáme kód jádra [9]:

```
git clone https://github.com/libos-nuse/net-next-nuse.git
cd net-next-nuse
git checkout libos-v4.4
```

Konfigurace jádra probíhá v adresáři net-next-nuse následujícím příkazem [9]:

make defconfig ARCH=lib

Nyní je třeba jádro zkompilovat [9].

make library ARCH=lib

Tím jsme dokončili kroky potřebné k získání upraveného jádra. Nyní je třeba jádro propojit s ns-3. V případě zachování cest, které jsou uvedeny výše k tomu slouží následující kroky.

Ve ~/dce/build/bin_dce. Změníme symbolický odkaz *liblinux.so* na nově zkompilované jádro takto:

Ln -s /home/student/net-next-nuse/arch/lib/tools/libsim-linux-4.4.0.so liblinux.so

Přejdeme do adresáře ~/dce/source /ns-3-dce a DCE nakonfigurujeme pro použití s novým jádrem:

./waf configure --with-ns3=$HOME/dce/build --enable-kernel-stack=$HOME/net-next-nuse/arch --prefix=$HOME/dce/build

Posledním krokem je nové sestavení DCE:

./waf build

2.6 Poznámka k použitým verzím ns-3

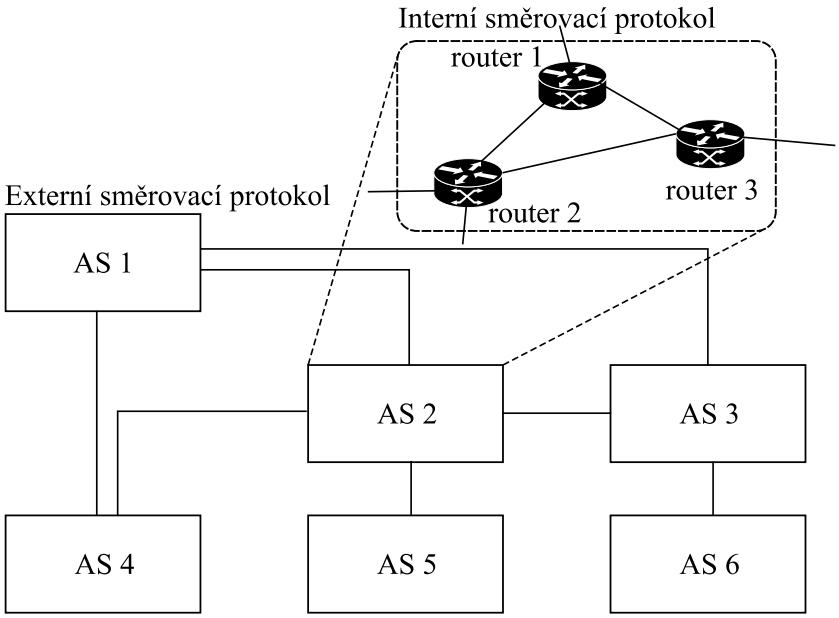
- Pro simulaci BGP je využito DCE s balíkem Quagga verze 1.8.
- TCP a UDP je simulováno v ns-3.23, funkčnost je ověřena i v ns-3.-21.
- Simulace SCTP je v DCE podporována od verze 1.3.
- Simulace síťových prvků a topologií funguje v ns-3.21 a výše.
- RIPv2 je podporován od verze ns-3.25.

3. AUTONOMNÍ SYSTÉMY

Internet jako takový je v dnešní době velmi rozsáhlý. Není tak možné, aby každý si uzel vyměňoval směrovací tabulky nebo udržoval kompletní mapu celé sítě, jak je tomu v případě protokolů RIP (Routing Information Protocol) či OSPF (Open Shortest Path First) [10] [11]. Tyto protokoly jsou určeny pouze pro směrování v rámci sítě jednoho autonomního systému a říká se jim IGP (Interior Gateway Protocols). Autonomní systém (dále jen AS) si můžeme představit jako síť, kterou spravuje jedna organizace a je v ní využívána jednotná politika směrování [12]. Je tak nutné vyřešit směrování mezi těmito AS. Směrovací protokol používaný mezi AS se označuje zkratkou BGP (Border Gateway Protocols).

3.1 Směrovací protokol BGP

BGP je v současné době používaným protokolem pro směrování mezi AS. Využívána je 4. verze tohoto protokolu, která je popsána v dokumentu RFC 4271. BGP je založeno na výměně směrovacích záznamů mezi sousedy. Každý směrovač si z přijatých záznamů vybere ten nejvhodnější, podle nastavené politiky a zařadí ho do své směrovací tabulky, ty potom šíří dalším sousedům [13]. Hlavní myšlenkou je vnímání sítě z hlediska AS, BGP tak považuje jednotlivé AS za základní jednotky a nezná vnitřní strukturu těchto sítí. Pouze eviduje adresy sítí, které AS obsahuje. Rozdíl je znázorněn na Obr. 3-1.



Obr. 3-1: Rozdíl mezi interními a externími směrovacími protokoly

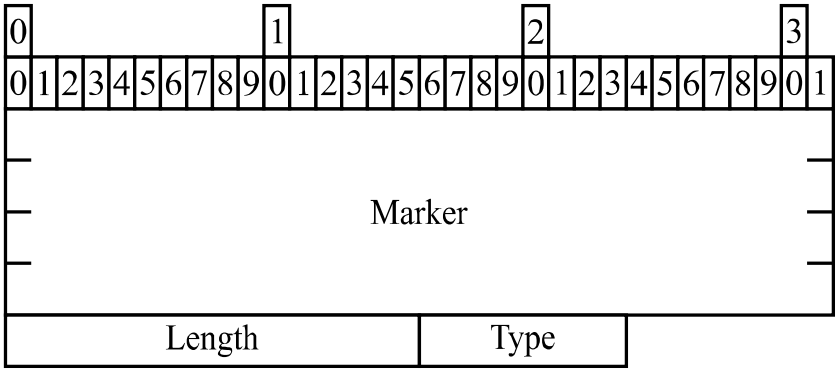
Paket je doručován pouze na hranici AS, hraničnímu směrovači. Další směrování už zajišťuje interní směrovací protokol. Je nutné si uvědomit, že vazby mezi AS nevznikají automaticky, ale je třeba je vždy nakonfigurovat ručně. Oproti jiným směrovacím protokolům je použito spolehlivého transportního protokolu TCP a portu 179 [14]. Jakmile si sousedé vytvoří TCP relaci, začnou si vyměňovat informace ve formě BGP zpráv, které jsou uvedeny v další kapitole.

3.2 BGP zprávy

Každá zpráva začíná hlavičkou a je následována obsahem zprávy. Nejmenší zprávu, kterou je možné zaslat, tvoří pouze hlavička bez datové části. Její velikost je tedy 19 B. Maximální velikost je omezena délkou 4096 B [15].

3.2.1 Hlavička BGP

Každou zprávu, která je zaslána, tvoří hlavička s pevnou velikostí 19 B. V závislosti na typu obsahuje zpráva i datovou část [15]. Na Obr. 3-2 je struktura hlavičky vyobrazena.

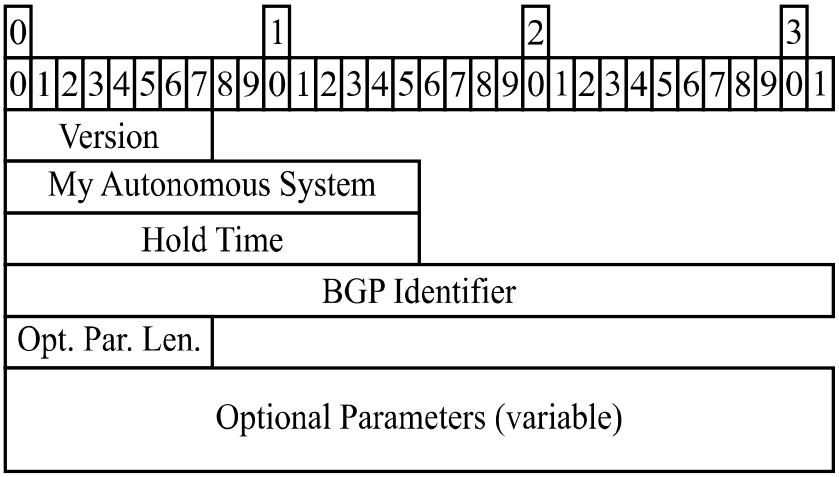


Obr. 3-2: Struktura hlavičky BGP zpráv

- **Marker** je 16 B pole, které musí být implementováno z důvodu zachování kompatibility s předchozími verzemi BGP. Všechny bity musí být nastaveny na binární jedničky [15].
- **Length** (délka) je pole o velikosti 2 B, které udává celkovou délku zprávy včetně záhlaví. Jak již bylo řečeno, minimální hodnota, kterou musí pole length udávat, je 19 B. Maximální pak 4096 B [15].
- **Type** (typ) je pole, které definuje typ zprávy. Jeho velikost je 1 B. Zprávy jsou následující [15]:
    1) OPEN
    2) UPDATE
    3) NOTIFICATION
    4) KEEPALIVE

3.2.2 Zpráva OPEN

První zprávou po sestavení TCP spojení, kterou si směrovače zasílají, je zpráva OPEN. V případě, že je zpráva doručena a akceptována, je potvrzena zprávou KEEPALIVE. Struktura datové části zprávy OPEN je vyobrazena na Obr. 3-3.



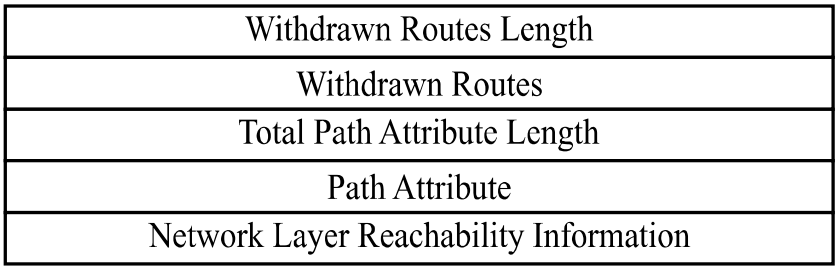
Obr. 3-3: Struktura datové části zprávy OPEN

Důležitá pole s vysvětlením:

- **Version** (verze) – 1 B. Označuje verzi protokolu. V současné době BGP verze 4 [15].
- **My Autonomous Systém** (můj autonomní systém) – 2 B. Číselné označení autonomního systému odesílatele [15].
- **Hold Time** (časovač) – 2 B. Hodnota udává informace pro výpočet času, který uplyne mezi zasláním další zprávy KEEPALIVE nebo UPDATE od odesílatele [15].

3.2.3 Zpráva UPDATE

Zprávou UPDATE jsou přenášeny směrovací informace mezi BGP směrovači. Jedná se tedy o velmi důležitou zprávu. Zpráva je přenášena vždy po vytvoření spojení pro prvotní konfiguraci. Následně až po změně v síti, neodesílá se periodicky [14]. Strukturu datové části zobrazuje Obr. 3-4.



Obr. 3-4: Struktura datové části zprávy UPDATE

- **Withdrawn Routes Length** (délka pole informací o neplatných routách) – 2 B. Indikuje velikost následujícího pole. Pokud je hodnota nastavená na 0 není přenášena žádná neplatná routa a následující pole není vůbec přenášeno [15].
- **Withdrawn Routes** (informace o neplatných routách) – Obsahuje seznam rout, do kterých nadále není možno směrovat pakety (například při výpadku) prostřednictvím odesílatele UPDATE zprávy. Pole má proměnlivou délku v závislosti na tom, kolik rout je třeba zneplatnit [15].
- **Total Path Attribute Length** (Celková délka pole atributy routy) – 2 B. délka následujícího pole. Může nabývat různých hodnot včetně nuly v případě, že se atributy nepřenáší [15].
- **Path Attribute** (atributy routy) – používá se k výběru nejvhodnější trasy v případě, že má směrovač více cest ke stejné síti. Pole má proměnnou délku [15].
- **NLRI** (informace o dostupnosti sítě) – pole má proměnnou délku a může obsahovat vice NLRI informací, pouze ale v případě, že pro všechny informace platí nastavené atributy. NLRI informace obsahuje [13]:
- **Length** (délka) – 1 B. Délka prefixu IP adresy.
- **Prefix** – začátek adresy cílové sítě.

3.2.4 Zpráva Notification

Zpráva notification je zaslána v případě zjištění chyby, následně je BGP spojení okamžitě ukončeno. Zpráva kromě hlavičky obsahuje také kód chyby (1 B), subkód chyby (1 B), který upřesňuje kde k chybě došlo a volitelná data [15].

3.2.5 Zpráva KeepAlive

Zprávy KeepAlive jsou zasílány pravidelně a slouží k informování o funkčnosti sousedního směrovače tak, aby nedošlo k vypršení časovače a tím zneplatnění směrovací informace [13]. Zpráva obsahuje jen hlavičku bez datové části a je tak vůbec nejkratší zasílanou zprávou. Maximální doba mezi zaslanými zprávami by měla být jedna třetina časovače Hold Time (přenášen ve zprávě Open) [15]. Ve výchozím nastavení bývá zpráva KeepAlive přenášena jednou za 60 sekund [12].

3.3 Směrovací proces

Kromě parametrů jako jsou rychlost linky, vzdálenost či počet skoků které známe z interních směrovacích protokolů vstupují do směrování v rámci AS další požadavky, a to především na ceně přenosu. Rozhodnutí o směrování tedy není vždy pouze na směrovacím systému, ale i na administrátorovi, který ovlivňuje nastavení u jím spravovaného autonomního systému [12].

Směrovače si kromě hlavní směrovací tabulky vytvářejí i několik dalších pomocných tabulek, které se nazývají RIB (The Routing Information Base). Tyto tabulky se skládají ze tří částí:

- **Adj-RIBs-IN** – kam si směrovací protokol ukládá směrovací informace z příchozích zpráv UPDATE, přijatých od ostatních BGP směrovačů. Do tabulky jsou ukládány informace o všech cestách, které jsou k dispozici. Tabulka je dále používána jako součást rozhodovacího procesu o tom, kudy bude směrováno [14].
- **Loc-RIB** – obsahuje místní směrovací informace pro použití lokálním směrovačem. Informace vybrány z tabulky Adj-RIBs-IN podle směrovací politiky [14].
- **Adj-RIBs-Out** – informace připravované pro odeslání k sousedovi zprávou UPDATE [14].

Z těchto tabulek je na základě zvolené politiky vybrána trasa pro směrování. K rozhodnutí jsou využívány parametry jako počet procházených AS (AS_PATH), případně administrátorem definované parametry váhy, místní preference (LOCAL_PREF) nebo výstupní diskriminátor (MED)

3.4 Shrnutí důležitých vlastností BGP

- Protokol TCP. Port 179.
- Směrování mezi autonomními systémy
- Routery, které mají navázáno TCP BGP spojení se označují za sousedy.
- Sousedy je nutno nakonfigurovat ručně.
- Každý AS má přiděleno jedinečné číslo pro identifikaci.
- O dostupnosti sousedů informuje pravidelně zasílaná zpráva KeepAlive.
- O směrování se rozhoduje podle cesty, směrovací politiky nebo dalších pravidel.

4. ÚLOHA 1: ZÁKLADNÍ KONFIGURACE BGP

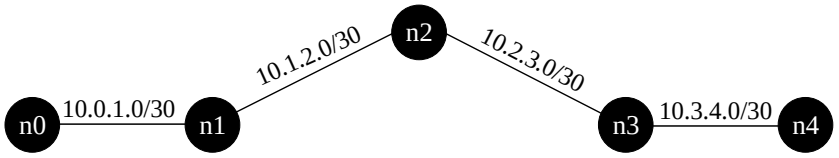
V této úloze bude použit simulátor ns-3 s podporou přímého vykonávání kódu DCE a s nainstalovaným balíkem Quagga, díky kterému je možné protokol BGP simulovat. Na topologii se budeme dívat z pohledu autonomních systémů, interní směrovací systémy řešit nebudeme. Ukážeme si, jakého portu a transportního protokolu je využito. Zaměříme se především na přenášené zprávy, které jsou uvedeny v teoretickém úvodu úlohy. Simulovat budeme i výpadek a následnou reakci směrovacího protokolu. Cestu sítí bude směrovací protokol určovat pomocí parametru AS_PATH, rozhodovat tedy bude celkový počet procházených autonomních systémů, což je v současné době jediná možnost, kterou ns-3 podporuje.

Pro psaní samotné simulace použijte jakýkoliv textový editor případně prostředí Eclipse. Spuštění bude probíhat přímo z terminálu.

Do virtuálního počítače si stáhněte předpřipravený soubor bgp_basic.cc tento soubor budeme dále upravovat pomocí následujícího návodu.

4.1 Základní topologie

V rámci první části úlohy bude vytvořena síť 5 BGP směrovačů. Každý tento směrovač bude reprezentovat samostatný autonomní systém. Topologie je zobrazena na Obr. 4-1.



Obr. 4-1: Úvodní topologie úlohy

4.1.1 Tvorba modelu

Do předpřipraveného souboru *bgp-basic.cc* postupně vkládejte kód na označená místa podle návodu v dalších kapitolách. Kód je vždy označen komentářem, který

souhlasí s komentářem ve výchozím souboru. Jako první je třeba vytvořit uzly, které jsou základním stavebním kamenem naší simulace. To se provede vložením následujícího kódu na začátek hlavní funkce *main*.

```
//vytvoreni uzlu
NodeContainer n;
n.Create (5);
```

Tímto jsme vytvořili kontejner pěti uzlů, následně je nutné definovat typ spojení. V této úloze bude požito jednoduchého spojení typu *pointToPoint*. K jeho vytvoření použijeme *PointToPointHelper*. Přenosová rychlost bude nastavena na hodnotu 1 Mbps, a aby nedocházelo k dodatečnému zpoždění, bude tato hodnota nastavena na 0ms.

```
//definice spoje
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("1Mb/s"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("0ms"));
```

Tímto jsme definovali parametry linky, ta však zatím nemá žádnou vazbu na síťové rozhraní uzlu. Vazby vytvoříme zadáním následujícího kódu.

```
//prirazeni spoje
NetDeviceContainer n0n1 = pointToPoint.Install (n.Get (0), n.Get (1));
NetDeviceContainer n1n2 = pointToPoint.Install (n.Get (1), n.Get (2));
NetDeviceContainer n2n3 = pointToPoint.Install (n.Get (2), n.Get (3));
NetDeviceContainer n3n4 = pointToPoint.Install (n.Get (3), n.Get (4));
```

Topologie už teď vypadá jako na obrázku. Nemáme však definované IP adresy na rozhraních uzlů. K přiřazení využijeme *Ipv4AddressHelper*, který udělá veškerou práci za nás. Potřeba je jen přiřadit adresní rozsah k lince, která uzly spojuje.

```
//nastaveni ip rozsahu
ipv4AddrHelper.SetBase ("10.0.1.0", "255.255.255.252 ");
      Ipv4InterfaceContainer i0i1 = ipv4AddrHelper.Assign (n0n1);
```

```
ipv4AddrHelper.SetBase ("10.1.2.0", "255.255.255.252");
       Ipv4InterfaceContainer i1i2 = ipv4AddrHelper.Assign (n1n2);
ipv4AddrHelper.SetBase ("10.2.3.0", "255.255.255.252");
      Ipv4InterfaceContainer i2i3 = ipv4AddrHelper.Assign (n2n3);
ipv4AddrHelper.SetBase ("10.3.4.0", "255.255.255.252");
      Ipv4InterfaceContainer i3i4 = ipv4AddrHelper.Assign (n3n4);
```

V prvním kroku dochází k definování IP rozsahu, následně je tento rozsah nainstalován na rozhraní spojené linkou. Adresy jsou přidělovány **postupně od počátku** rozsahu **v pořadí, v jakém byly uzly vytvořeny**.

Poté už se můžeme pustit k definici sousedů směrovacího protokolu BGP. K tomu nám poslouží *QuaggaHelper*. Nejprve je ale třeba povolit samotné směrování pomocí BGP na všech uzlech. K tomu slouží následující příkazy.

```
//zapnuti BGP pomoci QuggaHelper
QuaggaHelper quagga;
quagga.EnableBgp (n);
```

Jak bylo řečeno v úvodu, definice sousedů neprobíhá u BGP automaticky, ale je nutná ruční konfigurace. V ns-3 k tomu slouží funkce *BgpAddNeighbor* v následujícím tvaru.

```
quagga.BgpAddNeighbor(n.Get(vzchozi_uzel),"ip_souseda",quagga.GetAsn(n.Get(soused)))
```

Příkaz *GetAsn* vrací číslo autonomního systému, které je uzlům přiřazeno automaticky v pořadí, ve kterém jsou uzly vytvořeny. Sousedy je třeba definovat vždy pro oba uzly zvlášť. Pro zadanou topologii tedy probíhá definice sousedů následovně.

```
//BGP definice sousedu
//uzel0-uzel1
quagga.BgpAddNeighbor (n.Get (0), "10.0.1.2",quagga.GetAsn (n.Get (1)));
quagga.BgpAddNeighbor (n.Get (1), "10.0.1.1",quagga.GetAsn (n.Get (0)));
// uzel1-uzel2
quagga.BgpAddNeighbor (n.Get (1), "10.1.2.2",quagga.GetAsn (n.Get (2)));
quagga.BgpAddNeighbor (n.Get (2), "10.1.2.1",quagga.GetAsn (n.Get (1)));
```

```
// uzel2-uzel3
quagga.BgpAddNeighbor (n.Get (2), "10.2.3.2",quagga.GetAsn (n.Get (3)));
quagga.BgpAddNeighbor (n.Get (3), "10.2.3.1",quagga.GetAsn (n.Get (2)));
// uzel3-uzel4
quagga.BgpAddNeighbor (n.Get (3), "10.3.4.2",quagga.GetAsn (n.Get (4)));
quagga.BgpAddNeighbor (n.Get (4), "10.3.4.1",quagga.GetAsn (n.Get (3)));
```

Posledním krokem konfigurace BGP je instalace směrování na všechny uzly zadané topologie.

```
//BGP instalace
quagga.EnableZebraDebug (n);
quagga.Install (n);
```

4.1.2 Nastavení výstupů

Výsledky z této úlohy budou zobrazeny pomocí souborů pcap, které lze zobrazit v programu Wireshark. Vyexportovány také budou směrovací tabulky všech uzlů v různých časech. Pro kontrolu topologie je možné využít výstup do programu NetANIM, kterým je soubor ve formátu *xml*. Posledním výstupem je graf zatížení na vybraných linkách. Pokud neurčíme jinak, jsou všechny soubory uloženy do kořenového adresáře ns-3-DCE, tedy: ~/dce/source/ns-3-dce.

Začneme příkazy, kterými získáme soubory *pcap* pro každé rozhraní každého uzlu naší topologie.

```
//vystup pcap
pointToPoint.EnablePcapAll ("bgp-basic-node");
```

Za jméno souboru, v našem případě *bgp-basic-node*, budou přidána dvě čísla oddělená pomlčkou. První označuje číslo uzlu, druhé pak číslo rozhraní tohoto uzlu. Snadno tak identifikujeme soubory, které chceme zobrazit. K zobrazení souboru *pcap* je

možné využít příkazu *tcpdump* v terminálu. Uživatelsky přívětivější je ale zobrazení přímo v programu Wireshark.

Směrovací tabulky ve specifikovaném čase na všech uzlech je možné získat příkazem v následujícím tvaru. Příkaz nám do souboru uloží směrovací tabulky na všech uzlech každých 40 sekund.

```
	//smerovaci tabulky
	Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>
("bgp-basic-routes.log", std::ios::out);
	ipv4RoutingHelper.PrintRoutingTableAllEvery(Seconds(40),
routingStream);
```

Pro úplnost je zde uveden tvar příkazu, který uloží směrovací tabulky pro specifikovaný uzel. Tento kód kopírovat nemusíte.

```
ipv4RoutingHelper.PrintRoutingTableEvery(Seconds(40),n.Get(uzel),routingStream);
```

Vyexportovaný soubor je ve formátu *.log* a uložen pod jménem *bgp-basic-routes*. Pro zobrazení použijte obyčejný textový editor.

Pro zobrazení topologie v programu NetANIM, který umožňuje i animaci zasílaných dat, je nejdříve nutné definovat pohybový model a pozici uzlů. V našem případě se uzly pohybovat nebudou. Proto je nastavena funkce pro konstantní pozici. Pozice uzlů je možné definovat jednoduše v souřadnicovém systému x, y. Například takto.

```
	//NetANIM nastaveni polohy
	AnimationInterface::SetConstantPosition (n.Get (0), 10 , 10);
	AnimationInterface::SetConstantPosition (n.Get (1), 20 , 10);
	AnimationInterface::SetConstantPosition (n.Get (2), 30 , 0);
	AnimationInterface::SetConstantPosition (n.Get (3), 40 , 10);
	AnimationInterface::SetConstantPosition (n.Get (4), 50 , 10);
```

Následně už je jen potřeba zapnout ukládání a určit jméno souboru *xml*.

```
	//NetANIM ukladani
	AnimationInterface anim ("bgp-basic.xml");
	anim.EnablePacketMetadata(true);
```

Pro zobrazení grafu je před funkcí *main* definováno několik proměnných, jedna databáze a funkce pro ukládání dat. Hodnota proměnné *section* udává, jak často se budou data ukládat. Nejprve je tedy nutné se následujícími příkazy připojit na trasovací systém.

```
	//pripojeni k trasovacimu systemu
	Config::Connect("/NodeList/0/DeviceList/*/$ns3::PointToPointNetDevice/
PhyTxEnd",MakeCallback(&node0_phyTx));
```

Je třeba dát pozor, aby cesta k trasovacímu zdroji byla zkopírována do jednoho řádku. *PhyTxEnd* indikuje úspěšně přenesená data.

Pro ukládání dat je potřeba vytvořit cyklus, který volá funkci *saveData*, která je již vytvořena před hlavní funkcí *main*.

```
	//cyklus saveData
	for(double time=0; time<endTime; time=time+section)
		Simulator::Schedule (Seconds(time), &saveData, time);
```

Z uložených dat je poté možné vytvořit soubor ve formátu *.plt* následujícím kódem, který definuje formát zobrazení výsledného grafu. Kód je potřeba zkopírovat před řádek *return 0;*.

```
	//Nastaveni zobrazeni grafu
	string fileNameWithNoExtension = "bgp-basic";
	string graphicsFileName = fileNameWithNoExtension + ".png";
	string plotFileName = fileNameWithNoExtension + ".plt";
	string plotTitle = " Throughput";
	txTransferredDataset.SetTitle("n0n1");
	txTransferredDataset.SetStyle(Gnuplot2dDataset::LINES_POINTS);
	Gnuplot plot(graphicsFileName);
	plot.SetTitle("Throughput n0 a n1");
	plot.SetTerminal("png");
```

```
plot.SetLegend("Time [s]", " Throughput [kB/s]");
plot.AppendExtra ("set yrange [0:+0.6]");
plot.AddDataset(txTransferredDataset);
ofstream plotFile(plotFileName.c_str());
plot.GenerateOutput(plotFile);
plotFile.close();
system("gnuplot bgp-basic.plt");
```

4.1.3 Spuštění kódu

Hotový soubor *bgp-basic.cc* před spuštěním zkopírujeme do adresáře **~/dce/source/ns-3-dce/myscripts/ns-3-dce-quagga/example**. Následně je ještě nutné upravit soubor ***wscript***, který je umístěn v adresáři **~/dce/source/ns-3-dce/myscripts/ns-3-dce-quagga**, vložením tohoto kódu, který specifikuje použité moduly pro spuštění. **Kód vložte před řádek *def build(bld).***

```
module.add_example(needed=['core', 'dce-quagga', 'internet',  'netanim',
'network', 'point-to-point', 'mobility' ],
                            target='bin/bgp_basic',
                            source=['example/bgp_basic.cc'])
```

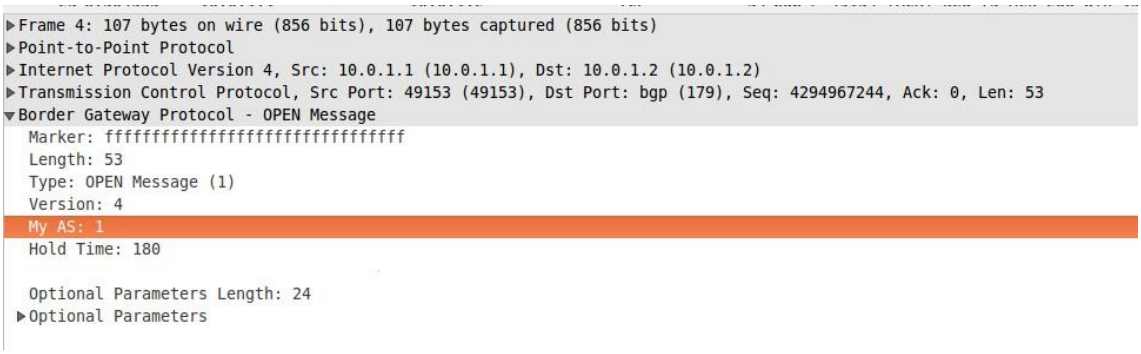
Teď se již můžeme přesunout k samotnému spuštění. V terminálu se přesuneme do adresáře, ve kterém se nachází dříve zmíněný soubor *wscript* a provedeme kontrolu sestavení programu příkazem: **./test.py -c core**. Pokud vše proběhlo v pořádku můžeme vytvořený program spustit příkazem: **./waf --run bgp_basic**. Pokud vše proběhlo bez chyb měli bychom v terminálu vidět výpis podobný tomu, který je na Obr. 4-2. **Doba trvání simulace je v tomto případě nastavena na 100 sekund.**

```
student@fekt:~/dce/source/ns-3-dce$ ./waf --run bgp_basic
Waf: Entering directory `/home/student/dce/source/ns-3-dce/build'
[ 11/403] lib/pkgconfig/libns3-dev-netlink-debug.pc:  -> build/lib/pkgconfig/libns3-dev-netlink-debug.
pc
[117/403] lib/pkgconfig/libns3-dev-dce-debug.pc:  -> build/lib/pkgconfig/libns3-dev-dce-debug.pc
[140/403] lib/pkgconfig/libns3-dev-dce-quagga-debug.pc:  -> build/myscripts/ns-3-dce-quagga/lib/pkgcon
fig/libns3-dev-dce-quagga-debug.pc
[141/403] cxx: myscripts/ns-3-dce-quagga/example/bgp_basic.cc -> build/myscripts/ns-3-dce-quagga/examp
le/bgp_basic.cc.17.o
[382/403] cxxprogram: build/myscripts/ns-3-dce-quagga/example/bgp_basic.cc.17.o -> build/myscripts/ns-
3-dce-quagga/bin/bgp_basic
Waf: Leaving directory `/home/student/dce/source/ns-3-dce/build'
'build' finished successfully (3.280s)
```

Obr. 4-2: Úspěšně dokončená simulace BGP

4.1.4 Zobrazení výsledků

V adresáři **~/dce/source/ns-3-dce** je teď uloženo osm souborů *pcap* pro každé rozhraní každého uzlu. Otevřeme si soubor *bgp-basic-node-0-0.pcap* a zobrazíme zprávu *OPEN*, která by měla vypadat podobně jako na Obr. 4-3.



Obr. 4-3: Zpráva OPEN zachycená na uzlu 0

Podívejte se, co je ve zprávě přenášeno a srovnejte s informacemi, které jsou uvedeny v úvodu úlohy. Podobně si zobrazte i další zprávy *KeepAlive* a *Update*.

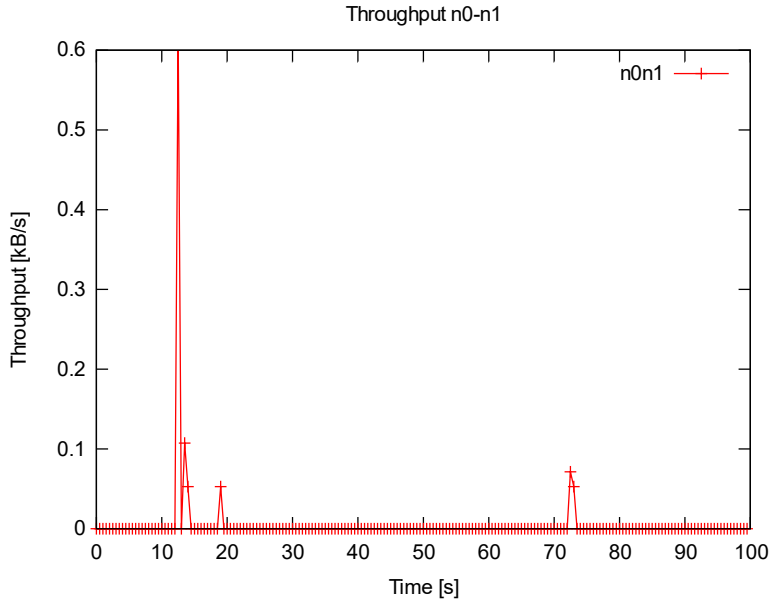
Nyní si zobrazíme směrovací tabulky, které jsou uloženy v souboru *bgp-basic-routes.log*. V případě správné funkčnosti by směrovací tabulka uzlu 2 měla vypadat jako na Obr. 4-4.

```
Time: 40s
Node: 2, Time: +40.0s, Local time: +40.0s, Ipv4StaticRouting table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
10.1.2.0        0.0.0.0         255.255.255.252 U     0      -      -   1
10.2.3.0        0.0.0.0         255.255.255.252 U     0      -      -   2
10.0.1.0        10.1.2.1        255.255.255.248 UGS   1      -      -   1
10.3.4.0        10.2.3.2        255.255.255.252 UGS   1      -      -   2
```

Obr. 4-4: Směrovací tabulka uzlu 2 v čase 40 sekund

Srovnejte tabulku se zadanou topologií a rozhodněte, zda směrování funguje korektně dle předpokladů.

Dalším výstupem je graf úspěšně přenesených dat v závislosti na čase, který je vynesen v souboru *bgp-basic.png*. Zobrazený průběh by měl vypadat jako graf na Obr. 4-5.



Obr. 4-5: Přenos dat mezi uzlem 0 a 1

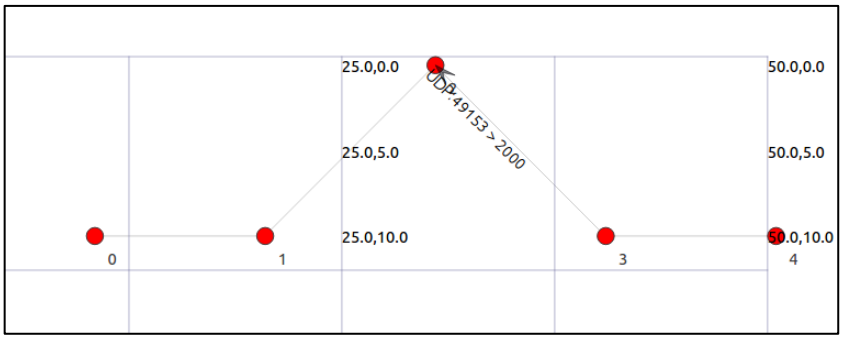
Topologii a přenášená data si můžete zobrazit i v programu NetANIM, ve kterém otevřete soubor *bgp-basic.xml*.

4.1.5 Otázky

- Jaký transportní protokol a jaký port BGP využívá?
- Do jaké vrstvy sítového modelu protokol BGP patří?
- Jak často je zasílána zpráva Update a KeepAlive?
- Vysvětlete nárůst v přenosu dat na začátku simulace, který je patrný ve výsledném grafu. Co představuje další nárůst ve zobrazeném průběhu?

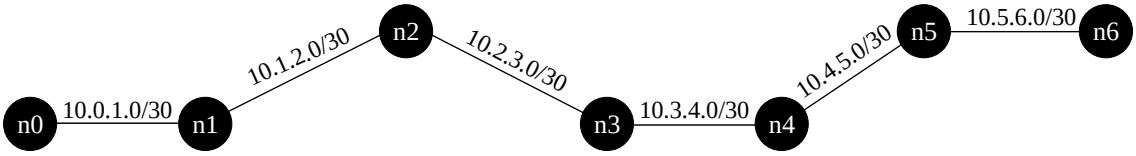
4.1.6 Samostatný úkol

V základním kódu je vytvořena a zakomentována jednoduchá UDP aplikace která zasílá data z uzlu 4 (klient) na uzel 1 (server). Aplikaci odkomentujte, simulaci znovu spusťte a ověřte, že směrování pomocí BGP opravdu probíhá správně. To můžete zjistit ze souborů *.pcap*, případně zobrazit v programu NetANIM, jehož výstup by měl vypadat zhruba jako na Obr. 4-6.



Obr. 4-6: Zobrazení přenosu v programu NetANIM

Samostatně přidejte do simulace dva další uzly tak, aby měla síť řetězový charakter jako na Obr. 4-7.

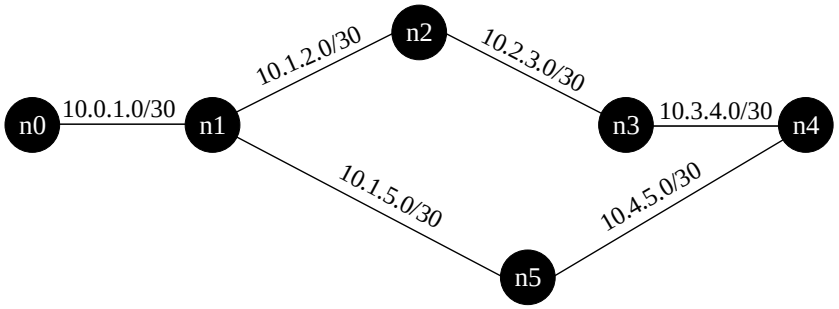


Obr. 4-7: Doplnění původní topologie

Kromě přidání uzlů, linek a definice BGP sousedů nezapomeňte také přidat pozici uzlů pro zobrazení v programu NetAnim. Upravte i původní UDP aplikaci tak, aby byl klient na uzlu 6. **Výsledky tohoto úkolu včetně zdrojového kódu si uložte pro pozdější kontrolu vyučujícím (například na plochu).**

4.2 Topologie s výpadkem

V této části se zaměříme na chování protokolu na výpadek linky mezi autonomními systémy. Vycházet budeme z původní topologie, kam k pěti uzlům přidáme jeden další uzel a spojíme ho s uzly 1 a 4. Topologie bude vypadat jako na Obr. 4-8.



Obr. 4-8: Topologie pro simulaci výpadku

V kódu vymažte uzly, které jste vytvořili v předchozím samostatném úkolu, zakomentujte aplikaci UDP a **délku simulace nastavte na 205 sekund** (editace

proměnné endTime). **Počet uzlů n nastavte na šest** a vytvořte nová rozhraní pro spojení mezi uzly:

```
NetDeviceContainer n1n5 = pointToPoint.Install (n.Get (1), n.Get (5));
NetDeviceContainer n4n5 = pointToPoint.Install (n.Get (4), n.Get (5));
```

Nastavte rozsahy IP:

```
ipv4AddrHelper.SetBase ("10.1.5.0", "255.255.255.252");
      Ipv4InterfaceContainer i1i5 = ipv4AddrHelper.Assign (n1n5);
ipv4AddrHelper.SetBase ("10.4.5.0", "255.255.255.252");
      Ipv4InterfaceContainer i4i5 = ipv4AddrHelper.Assign (n4n5);
```

Definujte sousedy:

```
//linka uzel1-uzel5
quagga.BgpAddNeighbor (n.Get (1), "10.1.5.2",quagga.GetAsn (n.Get (5)));
quagga.BgpAddNeighbor (n.Get (5), "10.1.5.1",quagga.GetAsn (n.Get (1)));
//linka uzel4-uzel5
quagga.BgpAddNeighbor (n.Get (4), "10.4.5.2",quagga.GetAsn (n.Get (5)));
quagga.BgpAddNeighbor (n.Get (5), "10.4.5.1",quagga.GetAsn (n.Get (4)));
```

A nastavte pozici uzlu n5 pro zobrazení v programu *NetANIM*.

```
AnimationInterface::SetConstantPosition (n.Get (5), 30 , 20);
```

Před nastavením výpadku je vhodné program spustit a případně opravit vzniklé chyby.

Výpadek budeme nastavovat mezi uzlem n1 a n5 pro jeho simulaci budeme využívat metodu pro změnu zpoždění na dané lince. Následující kód zkopírujte před hlavní funkci *main*.

```
//Zde vlozte metodu pro zmenu zpozdeni na lince mezi n1n5 pro simulaci vypadku
void ChangeDelay(){
Config::Set("/ChannelList/4/$ns3::PointToPointChannel/Delay",StringValue
("1000000ms"));}
```

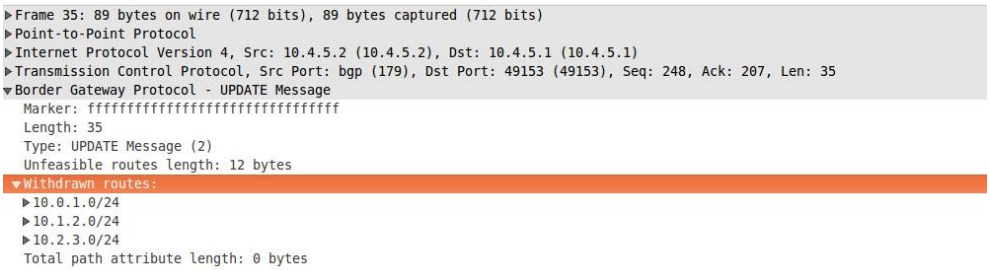
Číslování linek probíhá od nuly v pořadí, ve kterém jsou vytvořeny stejně, jako v případě uzlů. Věnujte tedy pozornost tomu, na jaké lince změnu zpoždění nastavujete. Hodnota zpoždění je nastavena na nesmyslně vysokou hodnotu tak, aby k výpadku došlo. K tomu, aby byla metoda provedena, musíme nastavit její volání v určitý čas pomocí následujícího příkazu, který vložíme před řádek spouštějící simulátor. Výpadek je nastaven na 50. sekundu po spuštění simulátoru.

```
//*Zde doplnte volani funkce pro zmenu zpozdeni*
Simulator::Schedule (Seconds(50), &ChangeDelay);
```

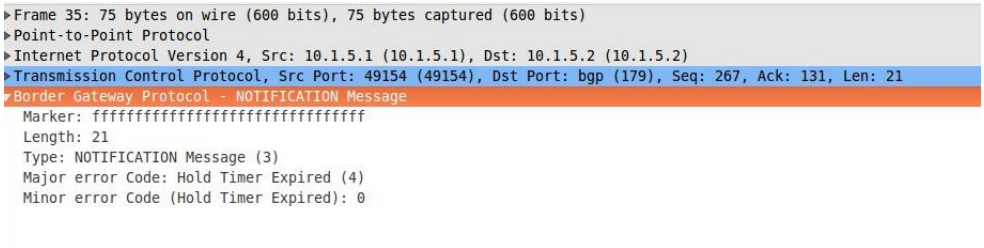
Nyní můžeme program opět spustit a prohlédnout si výsledky.

4.2.1 Zobrazení výsledků

Ve vytvořených souborech *.pcap* nalezněte zprávu UPDATE, která bude vypadat podobně jako na Obr. 4-9. Zajímavá je také zpráva NOTIFICATION, která je na Obr. 4-10.



Obr. 4-9: Zpráva UPDATE protokolu BGP



Obr. 4-10: Zpráva NOTIFICATION protokolu BGP

Prohlédněte si směrovací tabulky a zjistěte, kdy došlo k jejich změně, znovu také vytvořte graf a odpovězte na otázky uvedené v další kapitole.

4.2.2 Otázky

- Kdy směrovací systém zareagoval na výpadek? Proč právě v tuto chvíli?
- Jak se změna projevila ve směrovacích tabulkách?
- K čemu slouží zpráva NOTIFICATION?
- Co je obsahem zprávy UPDATE?
- Jak se výpadek projevil na zobrazeném grafu?

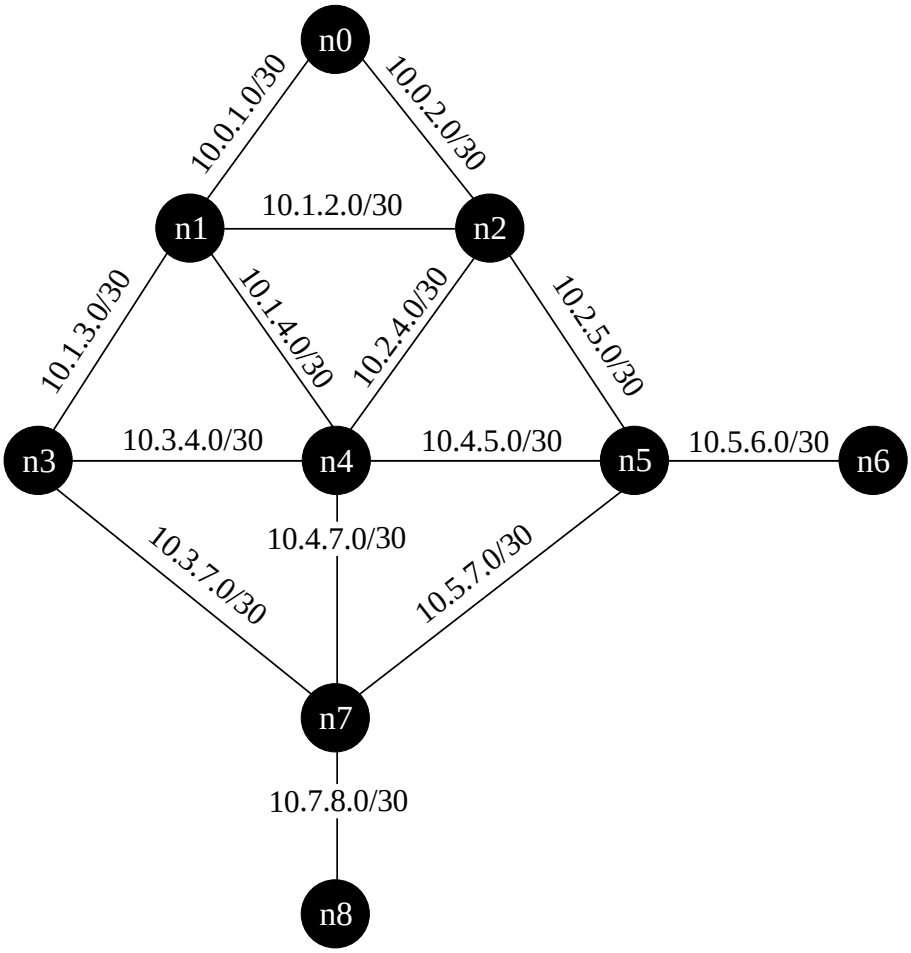
4.2.3 Samostatný úkol

Sami navrhněte novou metodu simulující obnovení linky mezi n1 a n5. Čas obnovení nastavte vhodně tak, aby k němu došlo až po reakci BGP na původní výpadek. **Nezapomeňte prodloužit délku trvání simulace, aby bylo možné výsledky zobrazit.** Vyberte si jeden ze souborů *.pcap,* na kterém bude nejlépe vidět, jak systém reaguje na výpadek. Okomentujte zasílané zprávy, výsledný graf a směrovací tabulky.

- Jak rychle je spojení obnoveno?

4.3 Samostatná topologie

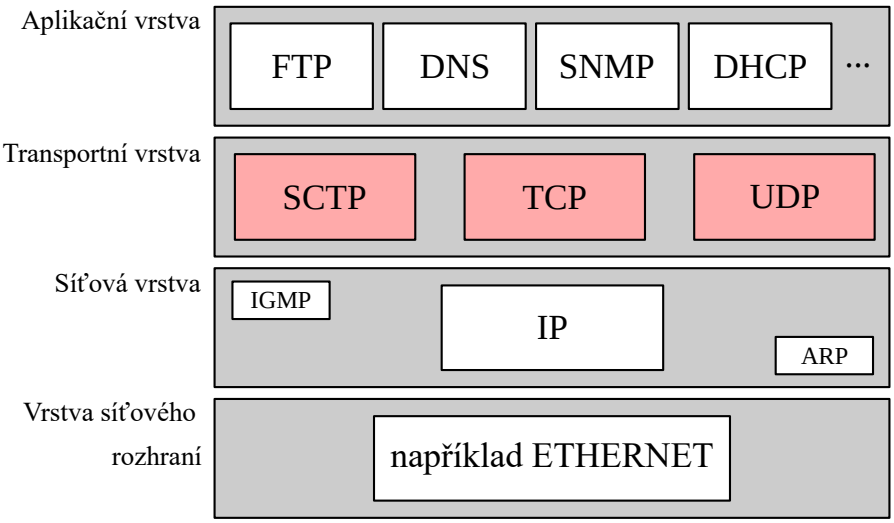
Do předem vytvořeného souboru *bgp_basic_sam.cc* doplňte konfiguraci BGP sousedů a pozici uzlů pro zobrazení v programu NetANIM podle topologie na Obr. 4-11. IP adresy jsou přiděleny podle čísel uzlů které linka spojuje. Druhý oktet adresy vždy obsahuje menší číslo. Pro linku z n0 do n1 je tak přidělená IP adresa z rozsahu 10.0.1.0/30. Před spuštěním simulace nezapomeňte na nutnost upravit soubor *wscript* tak, aby v něm byly uvedeny cesty a modely použité v nové simulaci. Pro kontrolu směrování vytvořte UDP aplikace mezi uzly n0 a n8 a uzly n1 a n6. Délku simulace nastavte na 80 sekund. **Výsledky připravte ke kontrole vyučujícím.**



Obr. 4-11: Topologie pro samostatnou práci

5. PROTOKOLY TRANSPORTNÍ VRSTVY

Tato kapitola je teoretickým úvodem k laboratorní úloze, která se týká protokolů, fungujících na transportní vrstvě. Blíže popsány budou protokoly UDP, TCP a novější protokol SCTP. Pozice těchto protokolů ve vrstvovém modelu TCP/IP je zobrazena na Obr. 5-1.



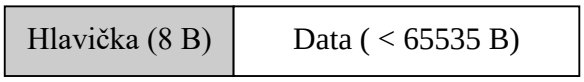
Obr. 5-1: Pozice transportních protokolů ve vrstvovém modelu TCP/IP [16]

5.1 UDP (User Datagram Protocol)

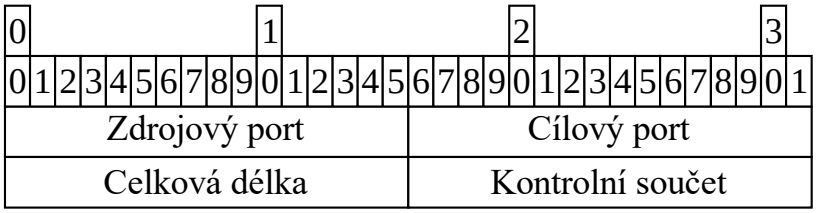
Protokol UDP zajišťuje nespojový a nespolehlivý přenos, z čehož plynou hlavní výhody i nevýhody. UDP nepodporuje žádný mechanismus řízení toku dat, a také nedochází k žádnému potvrzování při příjmu. Pouze zajišťuje určitou kontrolu chyb. V případě zjištěné chyby v přijímaném UDP datagramu, jak datovou jednotku nazýváme, je datagram jednoduše zahozen [16]. Všechny tyto nedostatky se ale stávají hlavní výhodou. UDP je velmi jednoduchý protokol s malou režií a minimálním zpožděním. Je tak vhodný zejména pro přenos malých zpráv a aplikace fungující v reálném čase [17].

5.1.1 Formát hlavičky

UDP datagram má pevnou délku hlavičky (8 bajtů) a proměnlivou délku datové části, což ilustruje Obr. 5-2. Formát datové časti je pak zobrazen na Obr. 5-3.



Obr. 5-2: Formát UDP datagramu



Obr. 5-3: Formát hlavičky UDP [18]

- **Zdrojový port** (Source port number) – číslo portu používané procesem běžícím na zdrojové straně komunikace. Délka je 16 bitů a může tak nabývat čísel od 0 do 65535. Když je zdrojem klient, který posílá žádost serveru, je číslo portu voleno podle požadavku procesu. V případě, že je zdrojem server, který odesílá odpověď, jedná se většinou o známé číslo portu [16] [18]. Některá z nich jsou uvedena v Tab. 5-1**.**
- **Cílový port** (Destination port number) – číslo portu využívané procesem běžícím na cílové straně. Délka je stejně jako v případě zdrojového portu 16 bitů. Pokud je cílem server, kterému klient odesílá požadavek, je číslo portu opět známé. V případě že je cílem klient, kterému server odesílá odpověď, je číslem portu ve většině případů dočasné číslo, které server zkopíruje z hlavičky doručené v datagramu žádosti [16] [18].
- **Celková délka** (Total length) – pole o velikosti 16 bitů které definuje celkovou délku datagramu, tedy datovou část včetně hlavičky. 16 bitů může definovat celkovou délku 0 až 65535 bajtů. Prakticky je však velikost mnohem menší, protože UDP datagram je zapouzdřen v IP datagramu, který má maximální velikost taktéž 65535 bajtů [16] [18].
- **Kontrolní součet** (Checksum) – pole použité pro detekci základních chyb v celém datagramu na transportní úrovni [16] [17].

5.1.2 Služby protokolu UDP

Základní služby, které protokol UDP poskytuje jsou uvedeny v následujícím textu.

- **Komunikace proces-proces** – pro komunikaci využívá protokol socketové adresy, což je kombinace IP adresy a portu [16]. Zvláště důležité je ale rozlišení na základě portů. Některé známé porty jsou uvedeny v Tab. 5-1.

Tab. 5-1: Vybrané známé porty

Číslo portu	Transportní protokol	Aplikační protokol
20	TCP	FTP – data
21	TCP	FTP – řízení
23	TCP	Telnet
25	TCP/UDP	SMTP
53	TCP/UDP	DNS
67	UDP	DHCP – server
68	UDP	DHCP – klient
80	TCP	HTTP
443	TCP/UDP	HTTPS

- **Nespojová forma komunikace** – každý datagram je přenášen samostatně bez závislosti na ostatních datagramech, a to i v případě, že pocházejí ze stejného procesu. Datagramy nejsou žádným způsobem číslovány. Neprobíhá navazování ani ukončování spojení. Každý datagram tak může putovat po jiné cestě [16].
- **Řízení toku dat** – kvůli své jednoduchosti nevyužívá UDP žádnou formu řízení toku dat [16]. Vysílač tak může zahltit přenosovou síť, případně i příjemce dat [17].
- **Kontrola chyb** – stejně jako v případě řízení toku dat neposkytuje, kromě kontrolního součtu, UDP kontrolu chyb. Odesílatel ani příjemce nemají žádné informace o tom, zda datagram dorazil, případně zda byla zpráva ztracena. V případě chyby zjištěné v kontrolním součtu je zpráva jednoduše zahozena. V případě nutnosti si tak kontrolu musí zajistit samotný proces [16].

- **Zapouzdřování a odpouzdřování dat** – služba vytváří jednotky transportní úrovně na straně odesílatele. Naopak na straně příjemce zbavuje datové jednotky záhlaví [17].
- **Vytváření front** – fronty jsou v UDP přímo spojeny s porty. V některých případech se vytváří odchozí i příchozí fronta. V jiných pouze příchozí, a to pro každý proces zvlášť [16].
- **Multiplexování a demultiplexování** – na straně odesílatele může fungovat několik procesů které chtějí posílat UDP datagramy. UDP přijímá zprávy z různých procesů rozdělených podle čísel portů, které multiplexuje. Podobně probíhá na straně příjemce demultiplexování, a po kontrole chyb a oddělení hlavičky předává data procesům podle čísla portu [16].

5.1.3 Shrnutí UDP

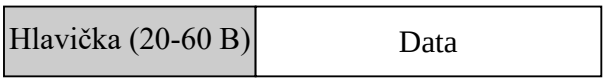
- Nespojový a nespolehlivý přenos.
- Malá režie přenosu, z toho plynoucí rychlé doručení.
- Datovou jednotu UDP na transportní vrstvě nazýváme datagram.
- Nemá mechanismy k řízení toku dat.
- Použití:
    - Jednoduché služby typu dotaz – odpověď. Například DNS.
    - Aplikace používající přenos v reálném čase. Například VoIP.

5.2 TCP (Transmission Control Protocol)

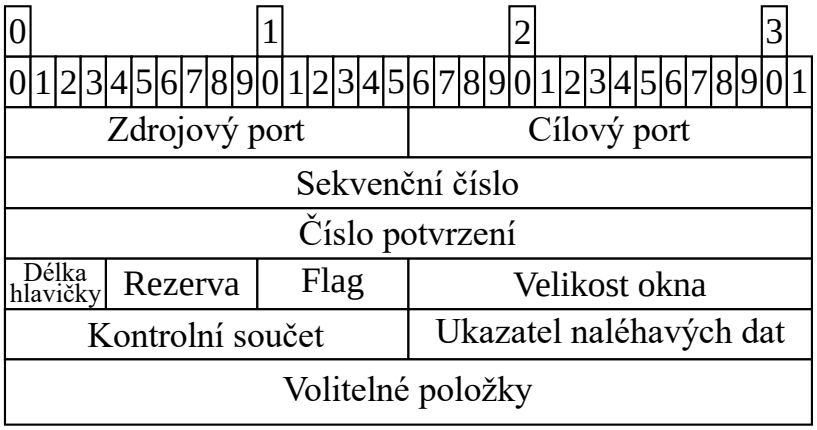
Oproti UDP je protokol TCP mnohem složitější. Komunikace má spojový charakter a protokol garantuje spolehlivé doručení ve správném pořadí. Protokol tak má mnohem větší režii a větší zpoždění. V případě UDP jsme mluvili o datagramech. Datagram je jednotka, u které není zajištěno doručení. To ale neplatí v případě TCP. Datová jednotka TCP je tedy označována jako segment [16].

5.2.1 Formát hlavičky

Kompletní segment se skládá z pole hlavičky a datové části. Segment je ilustrován na Obr. 5-4**.** Obě tato pole mají proměnlivou délku. Samotná hlavička (Obr. 5-5) má 20 až 60 bajtů a je mnohem složitější než hlavička UDP.



Obr. 5-4: Formát TCP segmentu



Obr. 5-5: Formát hlavičky TCP segmentu [19]

- **Zdrojový port** (Source port) – pole o velikosti 16 bitů. Stejně jako v případě UDP je v poli zapsán port procesu, který odesílá segmenty [19] [16]. Některé známé porty jsou uvedeny v Tab. 5-1.
- **Cílový port** (Destination port) – 16 bitů a opět stejný účel jako v případě UDP. Definuje port cílového procesu, který segmenty přijímá [19] [16].
- **Sekvenční číslo** (Sequence number) – 32bitové číslo které definuje pořadové číslo prvního bajtu v segmentu kterému náleží [19] [16].
- **Číslo potvrzení** (Acknowledgment number) – 32bitové číslo které slouží k potvrzení přijatých dat. Příjemce přičte k poslednímu správně přijatému bajtu jedničku a toto číslo v segmentu odešle jako potvrzení příjmu. Číslo potvrzení je běžně zasíláno společně s daty [19] [16].
- **Délka hlavičky** (Header length) – pole má velikost 4 bity a udává počet 4bajtových slov použitých v hlavičce. Minimální hodnota je 5 (20 bajtů) maximální pak 15 (60 bajtů) [19] [16].
- **Rezerva** (Reserved) – rezerva pro budoucí použití.

- **Příznakové bity** (Flags) – pole definuje šest různých stavů nebo příznaků, které mohou být kombinovány. Umožňují řízení toku, navázání a ukončení spojení a režim přenosu dat. Jejich význam je uveden dále [19] [16].
    - **URG** – naléhavá data obsažená v segmentu [16].
    - **ACK** – indikuje, že segment také potvrzuje data, která dříve přijala druhá strana [16].
    - **PSH** – indikuje okamžité předání dat aplikaci bez čekání [16].
    - **RST** – slouží k ukončení spojení (bez dalšího potvrzování) [16].
    - **SYN** – používáno při navazování spojení [16].
    - **FIN** – používáno při ukončování spojení odesílatelem [16].
- **Velikost okna** (Window size) – 16 bitů, které určují, kolik bajtů může vysílač přenést bez čekání na potvrzení. Velikost je obvykle určována přijímačem. Vysílač musí hodnotu respektovat [19] [16].
- **Kontrolní součet** (Checksum) – použití ke kontrole bezchybnosti, podobně jako v případě UDP. Na rozdíl od UDP je v případě TCP použití pole povinné [19] [16].
- **Ukazatel naléhavých dat** (Urgent pointer) – použito jen v případě, že je nastaven urgentní příznak (URG) [19] [16].
- **Volitelné položky záhlaví** (Options) – muže obsahovat až 40 bajtů volitelných informací [17].

5.2.2 Služby protokolu TCP

- **Komunikace proces-proces** – stejně jako v případě UDP probíhá komunikace mezi procesy na základě socketových adres [17].
- **Přenos toku dat** – proti UDP který používá pro přenos samostatné datagramy které jsou na sobě nezávislé, TCP vytváří proud bajtů [16]. Vytváří prostředí, ve kterém jsou přijímač i vysílač propojeny imaginárním okruhem. Díky číslování segmentů je potom možné data seskládat ve správném pořadí [17].
- **Plně duplexní přenos** – data mohou být odesílána současně oběma směry. Každý koncový bod pak disponuje vyrovnávací pamětí jak pro přijímané, tak odesílané segmenty [16] [17].

- **Multiplexování a demultiplexování** – stejně, jako v případě UDP, i TCP provádí multiplexování a demultiplexování na transportní vrstvě, a to ze vstupních a výstupních front oddělených podle aplikací [16] [17].
- **Spojově orientovaná služba** – na rozdíl od UDP je ke komunikaci nutné navázat spojení a vytvořit tak virtuální okruh [17]. Po navázání spojení probíhá obousměrná komunikace a následně je nutno spojení také ukončit [16].
- **Spolehlivý přenos dat** – TCP zaručuje spolehlivé doručení dat. K tomu používá potvrzování [16]. V případě nedoručení nebo chyby zajišťuje opětovný přenos.

5.2.3 Vlastnosti protokolu TCP

Pro poskytování služeb, které jsou výše uvedeny má TCP několik funkcí. Funkce jsou popsány dále.

- **Číslovací systém** – pro číslování segmentů je využíváno sekvenční číslo (Sequence number) a číslo potvrzení (Acknowledgment number). Tato čísla se nevážou k segmentů ale k bajtů. Číslo potvrzení vzniká z posledního přijatého bajtu a k němu připočtené jedničky. Strana tak nejen potvrzuje přijetí a správnost dat, ale také definuje číslo dalšího bajtu, které očekává. Příklad sekvenčních a potvrzovacích čísel je v Tab. 5-2. Vycházíme z předpokladu, že každý segment nese 512 bajtů [16] [19].

  Tab. 5-2: Příklad sekvenčních a potvrzovacích čísel [16]

Segment 1	Sekvenční číslo : 1025	Rozsah : 1025 - 1536
	Číslo potvrzení odeslané zpět : 1537	
Segment 2	Sekvenční číslo : 1537	Rozsah : 1537 - 2048
	Číslo potvrzení odeslané zpět : 2049	
Segment 3	Sekvenční číslo : 2049	Rozsah : 2049 - 2560
	Číslo potvrzení odeslané zpět : 2561	

- **Řízení toku dat** – je primárně založeno na velikosti okna viz. kapitola 5.2.1. Vysílač kontroluje, kolik dat je možné od procesu přijmout k odeslání. Podobně přijímací strana kontroluje, kolik dat muže vysílač odeslat a upravuje podle toho velikost okna [16] [19].

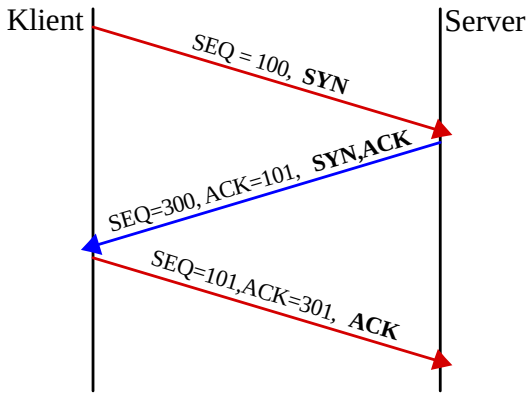
- **Kontrola chyb** – pro zajištění spolehlivost implementuje TCP mechanismy pro kontrolu chyb a správné reakce na ně. Chybou se rozumí poškození nebo úplná ztráta segmentu [16] [19].
- **Kontrola přetížení** – V případě UDP žádná kontrola přetížení neprobíhá a může tak snadno dojít k přetížení sítě. V TCP neprobíhá kontrola pouze přijímačem pomocí řízení toku, ale množství přenášených dat je také určeno úrovní přetížení sítě (tedy i od jiných protokolů) [16] [19].

5.2.4 Průběh komunikace

Jak již bylo řečeno v předchozích kapitolách, TCP je spojově orientovaný protokol. Před samotným přenosem dat je tedy třeba vytvořit virtuální okruh, a ten po dokončení přenosu také ukončit.

5.2.4.1 Navazování spojení

Proces navazování spojení se nazývá trojcestné potřesení rukou (three-way handshaking). Jeho průběh je zobrazen na Obr. 5-6 a dále rozebrán v textu.



Obr. 5-6: TCP navazování spojení [16]

- Klient odešle první segment s nastaveným příznakem SYN a sekvenční číslo nastaví na náhodnou hodnotu. V našem případě tedy 100. Tento segment nenese žádné potvrzovací čísla a ani informace o délce okna. Ty jsou přenášeny pouze v případě, že segment potvrzuje data, přijatá druhou stranou [17] .
- Server jako reakci odešle segment se dvěma příznaky SYN a ACK. SYN slouží pro nastavení sekvenčního čísla pro číslování bajtů odeslaných klientovi (300).

Příznakem ACK pak potvrzuje přijetí Segmentu SYN od klienta a nastavuje hodnotu dalšího očekávaného bajtu (+1). Segment nese potvrzení, proto je součástí i informace o velikosti okna [17] [19].

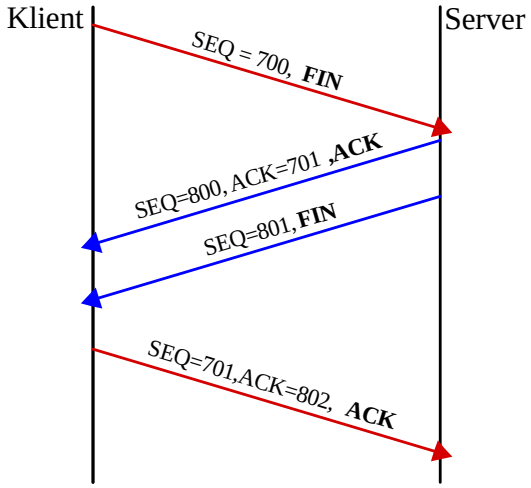
- Klient zasílá třetí segment v pořadí a dokončuje tak navazování spojení. Segment má nastavený příznak ACK a potvrzuje přijetí předchozího segmentu (SYN, ACK). Nastavení sekvenčních a potvrzovacích čísel funguje stejně jako u předchozího segmentu. Segment také nese informace o velikosti okna [17] [19].

5.2.4.2 Komunikace

Po navázání spojení může začít výměna zpráv. Komunikace může probíhat obousměrně a bajty každé zprávy jsou číslovány, aby bylo možné zaručit správné pořadí, a potvrzovány. K tomu slouží sekvenční a potvrzovací čísla, jejichž význam je uveden v kapitole 5.2.3.

5.2.4.3 Ukončování spojení

Po dokončení komunikace dochází k ukončení spojení, které je znázorněno na Obr. 5-7 a je velmi podobné navazování spojení. Nejobecnějším způsobem je čtyřcestné potřesení rukou (four-way handshake), kdy každá strana ukončuje spojení samostatně pomocí příznaku FIN. Je ale také možné využít zkrácenou trojcestnou variantu (three-way handshaking). V tomto případě jsou zprávy ACK a FIN ze serveru sloučeny do jedné [17] [19].



Obr. 5-7: TCP ukončování spojení [16]

5.2.5 Shrnutí TCP

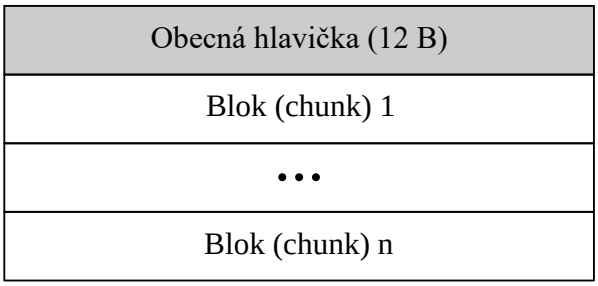
- Spojově orientovaný spolehlivý transportní protokol.
- Oproti UDP složitější protokol s větší režií. Nevýhodné zejména u malých objemů dat.
- Zaručuje doručení, a to ve správném pořadí.
- Datovou jednotku TCP na transportní vrstvě nazýváme segment.
- Zahrnuje řízení toku dat, řízení chyb a kontrolu přetížení.
- Použití v případě aplikací, které jsou citlivé na ztrátu dat jako FTP, HTTP, SMTP a další.

5.3 SCTP (Stream Control Transmission Protocol)

SCTP je novější protokol než TCP a UDP a má za cíl kombinovat vlastnosti těchto protokolů. Jedná se o spolehlivý transportní protokol orientovaný na přenos zpráv (podobně jako UDP). Poskytuje řízení přetížení a toku dat. Obsahuje mechanismy kontroly dat a zajišťuje spolehlivé doručení (podobně jako TCP). Hlavní vlastností je fakt, že po navázání spojení je možné přenášet řadu na sobě nezávislých proudů [16].

5.3.1 Formát hlavičky

Paket SCTP (Obr. 5-8) se skládá z obecné hlavičky a několika bloků (Chunk). Blok může sloužit k přenosu dat, případně může mít kontrolní funkci. Formát obecné hlavičky je zobrazen na Obr. 5-9 a informace, které přenáší jsou popsany níže.



Obr. 5-8: Formát SCTP paketu

- **Zdrojový port** (Source port address) – 16 bitů, které definují port na straně odesílatele [20].
- **Cílový port** (Destination port address) – 16 bitů, které definují port na straně příjemce [20].
- **Ověřovací značka** (Verification tag) – slouží k odlišení paketů, které byly zaslány při předchozím spojení [20].
- **Kontrolní součet** (Checksum) – Slouží ke stejným účelům jako kontrolní součet u TCP a UDP [20].

Obr. 5-9: Formát hlavičky SCTP paketu [20]

5.3.2 Blok (chunk)

Nese informace o tom, o jaký typ dat se jedná, dále pak příznakové bity, které mají různý význam podle typu přenášeného bloku a pole velikosti datového bloku. Typy bloků jsou uvedeny v Tab. 5-3. Příklad bloku, kterým jsou přenášena data je na Obr. 5-10. Vysvětlení polí je uvedeno v níže.

Obr. 5-10: Formát datového bloku (data chunk) [16]

Tab. 5-3: Typy bloků protokolu SCTP [16]

Typ	Datový blok (chunk)	Popis
0	DATA	Uživatelská data
1	INIT	Vytvoření asociace
2	INIT ACK	Potvrzení INIT bloku
3	SACK	Potvrzení
4	HEARTBEAT	Udržovací blok
5	HEARTBEAT ACK	Potvrzení HEARTBEAT
6	ABORD	Přerušení asociace (násilně)
7	SHUTDOWN	Žádost o ukončení asociace
8	SHUTDOWN ACK	Potvrzení SHUTDOWN
9	ERROR	Oznamuje chybu
10	COOKIE ECHO	K sestavení asociace
11	COOKIE ACK	Potvrzení COOKIE ECHO
14	SHUTDOWN COMPLETE	Ukončení asociace

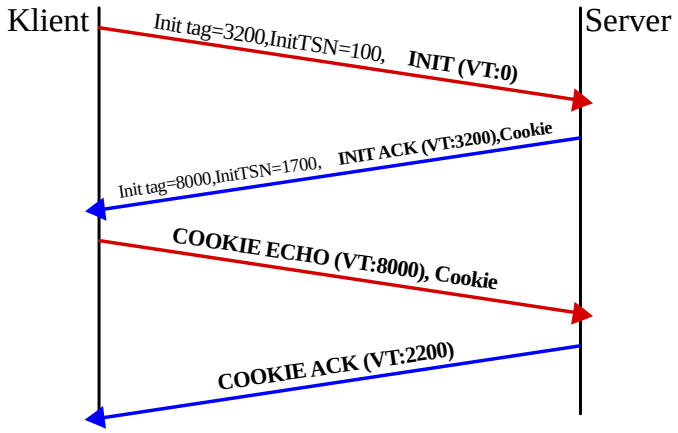
- **Typ (Type) –** v našem případě typ 0 pro datový blok **[16]**.
- **Příznakové bity** (Flag) – tři příznakové bity U, B a E. U je nastaveno na 1, pokud jde o neuspořádaná data. B a E fungují dohromady a označují pozici bloku ve fragmentované zprávě. Pokud je B=1 a E=1, zpráva není fragmentována. Pokud je B=1 a E=0, jde o první fragment. Pokud je B=0 a E=1, jde o poslední fragment a varianta B=0 a E=0 označuje střední fragment [16].
- **Velikost** (Length) – celková velikost bloku [16].
- **Sekvenční číslo přenosu** (TSN - Transmission sequence number) – 32 bitů. Je to pořadové číslo přenosu, které je nastaveno blokem *INIT* pro jeden směr přenosu a blokem *INIT ACK* pro opačný [16].
- **Identifikátor proudu** (SI - Stream identifier) – 16 bitů. Slouží k identifikaci proudu v rámci jedné asociace. To znamená, že všechny bloky, které náleží jednomu proudu mají stejný identifikátor [16].
- **Sekvenční číslo proudu** (SSN - Stream sequence number) – 16 bitů. Určuje pořadí bloku v rámci jednoho proudu [16].
- **Identifikátor protokolu** (Protocol identifier) – může být použit k definování typu dat aplikačním protokolem. SCTP toto pole ignoruje [16].

- **Uživatelská data** (User data) – nese samotná uživatelská data, pro která platí několik pravidel:
    - Datový blok musí nést data, která náleží pouze jedné zprávě [16].
    - Pole nemůže být prázdné [16].
    - Pokud data neskončí na hranici 32 bitů, je třeba přidat výplň. Výplň není zahrnuta v poli „Velikost“ [16].

5.3.3 Navazování a ukončování spojení

SCTP je stejně jako TCP spojově orientovaný protokol. Při navazování spojení je tedy vytvořen komunikační kanál, který je následně také nutné ukončit. Pro spojení se v případě SCTP používá termín asociace [20].

Pro sestavení asociace využívá protokol čtyřcestné potřesení rukou (four-way handshake). Asociace je zobrazena na Obr. 5-11 a popsána níže.

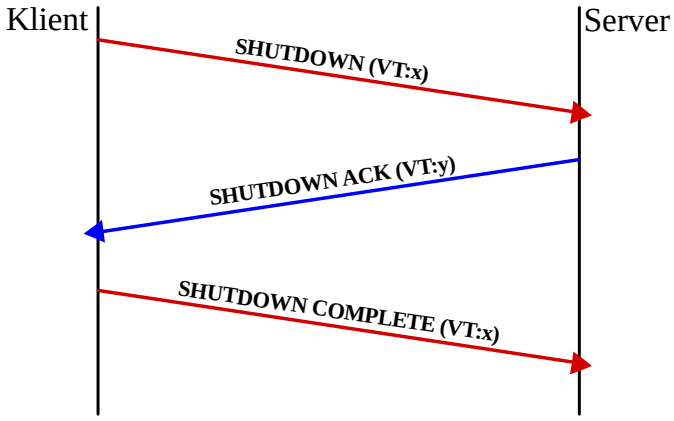


Obr. 5-11: SCTP vytváření asociace [16]

- Klient začíná navazování asociace prvním paketem, který obsahuje blok INIT. Kromě sekvenčního čísla přenosu (*TSN*), které je popsáno v kapitole 5.3.5, nese také inicializační značku (*Init tag)*, která slouží opačné straně komunikace k označení paketů (*verification tag-VT*), a nastavenou velikost okna [16].
- Server odpovídá zprávou INIT ACK, která opět nese hodnoty *Init tag*, *TSN*. K těmto informacím přidá zprávu *Cookie*. Ta definuje stav serveru. Opět je přítomna informace o velikosti okna [16].
- Klient odešle *COOKIE ECHO*, do kterého beze změny zkopíruje zprávu *COOKIE*, doručenou serverem [16].

- K dokončení sestavení spojení odešle server *COOKIE ACK*, což je potvrzení o přijetí předchozí zprávy od klienta [16].

Po vytvoření asociace může probíhat obousměrný přenos dat a po jeho dokončení je nutné asociaci ukončit. Ukončení asociace je na Obr. 5-12.



Obr. 5-12: SCTP ukončení asociace [16]

- Klient odešle požadavek na ukončení spojení *SHUTDOWN* [16].
- Ten server akceptuje a ukončí přijímání dat od procesu (pokud má ve frontě neodeslaná data, odešle je i po přijetí *SHUTDOWN).* Po odeslání všech zbylých dat (v případě že taková data jsou) zasílá potvrzení *SHUTDOWN ACK* [16].
- Asociaci ukončuje klient zprávou *SHUTDOWN COMPLETE* [16].

5.3.4 Služby protokolu SCTP

- **Komunikace proces-proces** – v podstatě stejné jako u TCP, využívá také stejné porty jako TCP, k nim přidává i další speciální porty využívané protokolem SCTP [16] [20].
- **Víceproudá komunikace** – v případě TCP je mezi klientem a serverem vytvářen jediný komunikační proud. Pokud dojde ke ztrátě nebo chybě v kterékoliv části proudu, je následně blokován i zbytek dat. To je problém hlavně hlavně když potřebujeme využít služeb v reálném čase. V případě SCTP může být využito několik proudů. Chyba, která se vyskytne v některém z proudů, neovlivňuje přenos v ostatních [16] [20].
- **Multihoming** – v případě, že některá ze stran disponuje více adresami, SCTP je může použít pro přenos. Když selže jedno rozhraní nebo cesta je možné využít

jiné rozhraní, a tím zajistit přenos bez přerušení. Spojení je tak odolnější než v případě TCP [16] [20].

- **Plně duplexní přenos** – stejně jako v případě TCP mohou data proudit oběma směry [16] [20].
- **Spojově orientovaná služba** – podobně jako u TCP. Spojení se v případě SCTP nazývá asociace [16] [20].
- **Spolehlivý přenos dat** – SCTP používá mechanismy potvrzení, čímž zaručuje spolehlivý přenos [16] [20].

5.3.5 Vlastnosti protokolu SCTP

- **Pořadové číslo přenosu** – v rámci TCP se číslují bajty, v případě SCTP dochází k číslování datových bloků (*data chunk*). Pořadové číslo je 32 bitů dlouhé a může nabývat hodnot od 0 až k (2^{32}-1). Toto číslo musí každý datový blok obsahovat [16].
- **Identifikátor proudu** – v každé SCTP asociaci může proudit několik toků dat. Každý proud je identifikován 16bitovým číslem začínajícím od nuly. Díky tomu může být proud identifikován a řádně zpracován [16].
- **Sekvenční číslo proudu** – každý datový blok, který je doručen, je přiřazen příslušnému proudu na základě identifikátoru proudu. K zajištění správného pořadí bloků je využito sekvenčního čísla na úrovni každého proudu [16].
- **Paket** – paket v případě SCTP můžeme přirovnat k segmentu TCP. Každý paket nese informace potřebné k doručení a muže do něj být zabaleno několik datových bloků (*chunk*) [16].
- **Číslo potvrzení** – v rámci SCTP jsou potvrzovány bloky dat. Potvrzování je vázáno k pořadovému číslu přenosu [16].
- **Řízení toku dat** – Stejně jako v případě TCP je implementováno řízení toku dat tak, aby nedošlo k přetížení přijímače nebo linky [16].
- **Kontrola chyb** – ke kontrole chyb se používají pořadová a potvrzovací čísla. Tím je zajištěna spolehlivost. [16].
- **Kontrola přetížení** – SCTP řídí kolik datových bloků může odeslat do sítě. Podobně jako u TCP [16].

5.3.6 Shrnutí SCTP

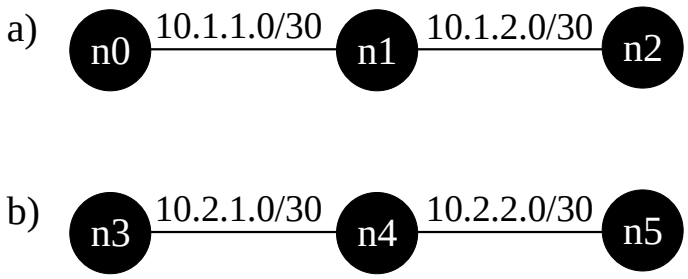
- Spolehlivý spojově orientovaný transportní protokol.
- Spojení SCTP se nazývá asociace.
- Kombinuje některé vlastnosti TCP a UDP.
- Zahrnuje řízení toku dat, řízení chyb a kontrolu přetížení
- Podporuje přenos ve více proudech, které na sobě nejsou závislé.
- Podporuje MULTIHOMING.
- Přenášenou jednotkou je paket. Ten obsahuje jednotlivé datové bloky (*chunk*).
- Protokol se používá zejména v oblasti telekomunikací. Například sada signalizačních protokolů SIGTRAN využívá pro přenos právě SCTP.

6. ÚLOHA 2: SROVNÁNÍ TRANSPORTÍCH PROTOKOLŮ UDP, TCP A SCTP

V laboratorní úloze se zaměříme na srovnání protokolů TCP a UDP. Ukážeme si, jak probíhá navazování spojení. Reakci na chybovost doručování v síti, chování protokolů v případě výpadku a přetížení. V závěru si také ukážeme přenos dat pomocí protokolu SCTP.

6.1 Základní topologie pro srovnání TCP a UDP

V první části budeme simulovat průběh komunikace TCP a UDP ve vlastních, na sobě nezávislých sítích. Zapojení je vyobrazeno na Obr. 6-1.



Obr. 6-1: Základní topologie úlohy a) TCP b) UDP

6.1.1 Tvorba modelu

Do souboru *TCP_UDP_vychozi.cc* budeme postupně vkládat kód na označená místa. K editaci můžete použít jakýkoliv textový editor případně prostředí *Eclipse*. Postupně nám tak v první části vzniknou jednoduché sítě, na kterých bude probíhat komunikace z uzlu n0 na n2 a z uzlu n3 a n5. Nejprve je nutné vytvořit uzly, které budou reprezentovat koncové stanice odesílající/přijímající data a směrovač mezi nimi. Z topologie sítě je patrné, že potřebujeme vytvořit šest uzlů což uděláme vložením následujícího kódu:

```
//Vytvoreni uzlu
NodeContainer n;
n.Create (6);
```

Uzlům je třeba doplnit vazby tak, aby odpovídaly sítím na obrázku. Následující kód ukazuje vytvoření kontejneru uzlů sítě pro komunikaci pomocí TCP:

```
//Vazba mezi uzly
//TCP
NodeContainer n0n1 = NodeContainer (n.Get (0), n.Get (1));
NodeContainer n1n2 = NodeContainer (n.Get (1), n.Get (2));
```

Samostatně vytvořte kontejnery pro síť, na které poběží UDP. Respektujte název kontejnerů, v případě UDP n3n4 a n4n5. Kontejnery vytvořte na místě označeném //UDP.

Dále definuje linku typu point-to-point, kterou později použijeme k propojení uzlů. V této části také nastavíme přenosovou rychlost a zpoždění. Přenosová rychlost bude nastavena na 1Mb/s, zpoždění potom na 2ms.

```
//Definice linek p2p
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate",DataRateValue(DataRate("1Mb/s")));
p2p.SetChannelAttribute("Delay",TimeValue(MilliSeconds(1)));
```

Point-to-point linku následně přiřadíme ke dříve vytvořeným kontejnerům, které obsahují uzly. Pro první síť je přiřazení uvedeno v následujícím kódu.

```
//Point-to-point
//TCP
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d1d2 = p2p.Install (n1n2);
```

Samostatně opět doplňte přiřazení pro druhou síť. *NetDeviceContainer* d3d4 a d4d5.

V dalším kroku je třeba na všechny uzly nainstalovat protokolový zásobník následujícími příkazy.

```
//Internet Stack (protokolovy zasobnik)
InternetStackHelper internet;
internet.Install (n);
```

Nyní pomocí Ipv4AddressHelperu přidělíme rozsahy IP adres. Pro adresy použijeme prefix /30, který odpovídá masce 255.255.255.252. Adresy totiž přiřazujeme ke spojení bod-bod, tedy dvěma rozhraním.

```
//Prirazeni adresnich rozsahu
Ipv4AddressHelper ipv4;
//TCP
ipv4.SetBase ("10.1.1.0", "255.255.255.252");
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);
ipv4.SetBase ("10.1.2.0", "255.255.255.252");
Ipv4InterfaceContainer i1i2 = ipv4.Assign (d1d2);
//UDP
ipv4.SetBase ("10.2.1.0", "255.255.255.252");
Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);
ipv4.SetBase ("10.2.2.0", "255.255.255.252");
Ipv4InterfaceContainer i4i5 = ipv4.Assign (d4d5);
```

Aby bylo možné v síti komunikovat, musíme povolit směrování. To nám v ns3 zajišťuje *Ipv4GlobalRoutingHelper*. Směrování povolíme tímto jednoduchým příkazem:

```
//Povoleni smerovani v siti
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Tímto jsme dokončili konfiguraci sítě. V další kapitole budeme vytvářet jednotlivé aplikace.

6.1.2 Tvorba aplikací

Ve síti vytvoříme samostatnou aplikaci. Aplikaci TCP, která bude zasílat data z uzlu n0 na uzel n2, a aplikaci UDP která bude zasílat data z uzlu n3 na n5. V první řadě si definujeme několik proměnných pro porty aplikací (port 9, na tomto portu jsou data po přijetí zahozena) a také množství přenesených dat v bajtech. Množství je pro obě aplikace stejné, tak, aby bylo srovnávání jednodušší. Obě aplikace jsou spuštěny v čase 1 sekunda od spuštění simulace, ukončeny pak v čase 9 sekund.

```
//Vytvoreni aplikaci
uint32_t maxBytes = 102400;   //Celkova velikost prenesenych dat
uint16_t portTCP = 9;         // Discard port (RFC 863)
uint16_t portUDP = 9;         // Discard port (RFC 863)
```

Definice aplikace TCP vypadá následovně:

```
      //TCP n0 - n2
      //n0 odesilani
      OnOffHelper onoffTCP ("ns3::TcpSocketFactory",Address
(InetSocketAddress (i1i2.GetAddress (1), portTCP)));
      onoffTCP.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
      onoffTCP.SetConstantRate (DataRate ("0.5Mb/s"));
      onoffTCP.SetAttribute ("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
      ApplicationContainer appsTCP = onoffTCP.Install (n.Get (0));
      appsTCP.Start (Seconds (1.0));
      appsTCP.Stop (Seconds (9.0));
      //n2 prijimani
      PacketSinkHelper sinkTCP ("ns3::TcpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), portTCP)));
      appsTCP = sinkTCP.Install (n.Get (2));
      appsTCP.Start (Seconds (1.0));
```

Pro UDP aplikaci potom:

```
      //UDP n3 - n5
      //n3 odesilani
      OnOffHelper onoffUDP ("ns3::UdpSocketFactory",Address
(InetSocketAddress (i4i5.GetAddress (1), portUDP)));
      onoffUDP.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
      onoffUDP.SetConstantRate (DataRate ("0.5Mb/s"));
      ApplicationContainer appsUDP = onoffUDP.Install (n.Get (3));
      appsUDP.Start (Seconds (1.0));
      appsUDP.Stop (Seconds (9.0));
      //n5 prijimani
      PacketSinkHelper sinkUDP ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), portUDP)));
      appsUDP = sinkUDP.Install (n.Get (5));
      appsUDP.Start (Seconds (1.0));
```

6.1.3 Nastavení výstupů a spuštění

Pro kontrolu správné funkce nastavíme na všech rozhraních zachytávání komunikace, které je následně možné otevřít v programu *Wireshark*. Uděláme to následujícím příkazem:

```
      //Zachytavani Pcap
      p2p.EnablePcapAll ("tcp_udp");
```

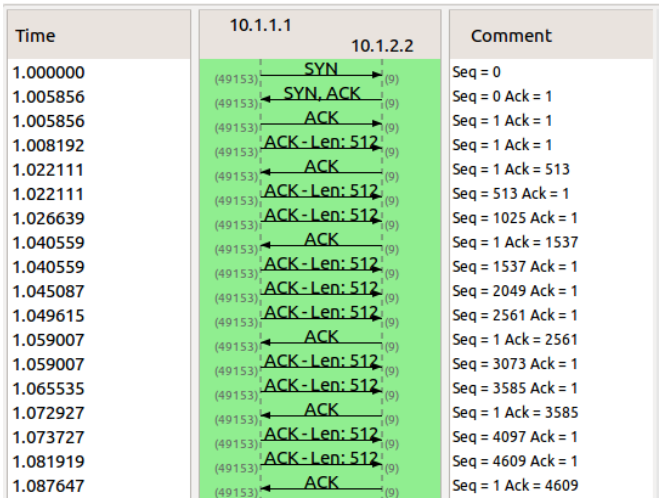
Poslední nutnou položkou jsou příkazy pro spuštění simulátoru. Délka trvání simulace je prozatím nastavena na 10 sekund.

```
//Spusteni simulatoru
Simulator::Stop (Seconds (10));
Simulator::Run ();
Simulator::Destroy ();
```

6.1.4 Spuštění simulátoru a zobrazení výsledků

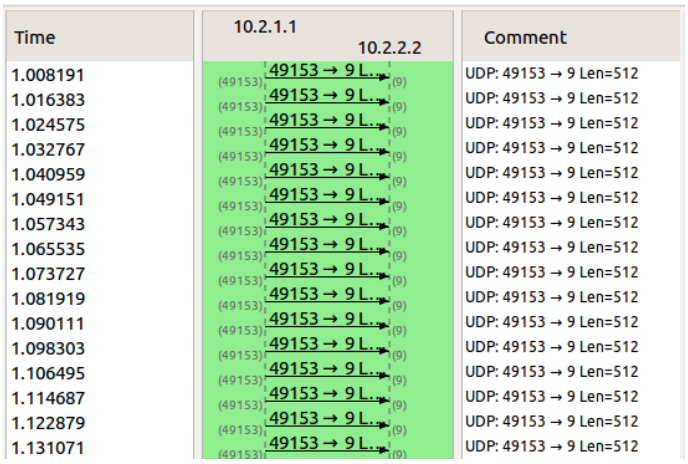
Námi vytvořený soubor zkopírujeme do adresáře *scratch*, který se nachází v domovském adresáři ns-3 (**~/ns-allinone-3.23/ns-3.23/scratch**). V terminálu se přesuneme do adresáře ns-3 (**~/ns-allinone-3.23/ns-3.23**) a simulaci spustíme příkazem **./waf --run *TCP_UDP_vychozi***. Pokud vše proběhlo bez chyb, měli byste na konci výstupu z terminálu vidět hlášku *„'build' finished successfully.“*

Správnost ověříme ze souborů *.pcap*, zachycených na každém rozhraní, které jsou uloženy v adresáři ns-3. Jména souborů odpovídají námi zadanému jménu *tcp_udp_vychozi-číslo_uzlu-číslo_rozhraní.pcap*. Otevřete si soubor zachycený na uzlu n0, na kterém probíhá komunikace TCP. V záložce *Statistics/Flow Graph/TCP Flows* si zobrazte komunikaci, která by měla vypadat jako na Obr. 6-2. Srovnejte s poznatky z teoretického úvodu.



Obr. 6-2: TCP komunikace zachycená na uzlu n0 – Flow graph

Stejně tak (*Statistics/Flow Graph/All Flows*) si otevřete soubor zachycený na uzlu n3, na kterém probíhá komunikace pomocí UDP (Obr. 6-3).



Obr. 6-3: UDP komunikace zachycená na uzlu n3 – Flow graph

6.1.4.1 Doplňující otázky TCP

- Zaměřte se na navazování spojení a na sekvenční a potvrzovací čísla. Funguje vše podle očekávání?
- Na konci přenosu si všimněte ukončování spojení. Jde o tří nebo čtyřcestné potřesení rukou?
- Jaká je velikost datové části v případě TCP?

6.1.4.2 Doplňující otázky UDP

- V čem vidíte hlavní rozdíl oproti TCP, který je patrný ze zachycené komunikace?
- Jaká je velikost datové části v případě UDP?

6.1.5 Doplnění simulace o měření parametrů a grafické zobrazení

Nyní doplníme simulaci o grafy přijatých a odeslaných dat. Také si doplníme části kódu, které nám zprostředkují měření zpoždění, ztrátovosti a dalších užitečných parametrů. Pro měření je použita třída *Database*, která je uložena v samostatném souboru *DatabaseTCP_UDP.txt*, a je třeba ji zkopírovat **před hlavní funkci *main***. Součástí

souboru je také několik funkcí, které se volají v případě, že dojde k nastaveným událostem (v našem případě nejčastěji odeslaný/přijatý paket zahození a podobně). Funkce jsou významněji použity v dalších částech úlohy.

Vytvořené databáze je třeba naplnit daty. K tomu využijeme trasovací systém ns-3. V případě odeslání či doručení paketu aplikací bude volána funkce *sendPacket* nebo *receivePaket*. To zařídíme následujícími příkazy, které je potřeba vložit na označené místo.

```
        //Trasovani aplikace
        Config::Connect("/NodeList/0/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "tcp"));
        Config::Connect("/NodeList/2/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "tcp"));

        Config::Connect("/NodeList/3/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "udp"));
        Config::Connect("/NodeList/5/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "udp"));
```

Také si budeme chtít zobrazit graf odeslaných dat. K tomu je potřeba vložit tyto příkazy:

```
        //Trasovani zatizeni-odesilaci strana
        Config::Connect("/NodeList/0/$ns3::Ipv4L3Protocol/Tx",MakeBoundCallback
(&trasmitPacket,"tcp"));
        Config::Connect("/NodeList/3/$ns3::Ipv4L3Protocol/Tx",MakeBoundCallback
(&trasmitPacket,"udp"));
```

A následně zavolat funkci pro ukládání v čase kdy jsou spuštěny aplikace:

```
        //Volani funkce pro mereni zatizeni
        Simulator::Schedule(Seconds(1), &saveThroughput);
```

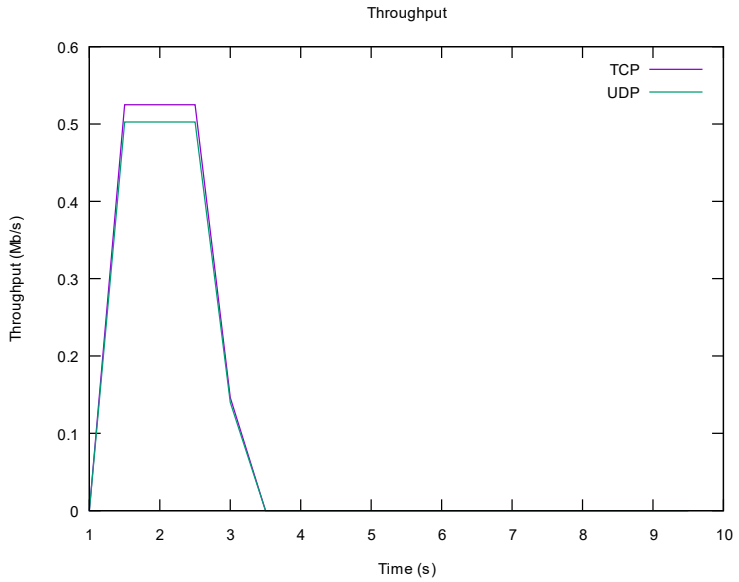
Nyní si z dat, které jsme zaznamenali, vypíšeme výsledky do terminálu. K tomu jsou na konci výchozího souboru umístěny zakomentované příkazy označené jako //Vystup terminal a //Spolecny graf. Ty nyní odkomentujte (položky "Number of dropped packets tcp/udp: " prozatím nechte zakomentované).

6.1.6 Zobrazení výsledků

Nyní simulátor opět spustíme známými příkazy v terminálu, a pokud vše proběhne v pořádku, dostaneme na výstupu informace, které vypadají takto:

```
TCP data sent: 102400 B
UDP data sent: 102400 B
TCP data received: 102400 B, error rate: 0 %
UDP data received: 102400 B, error rate: 0 %
Average latency for TCP: 11.1652 ms
Maximum latency for TCP: 18.848 ms
Average jitter for TCP: 0.127196 ms
Average latency for UDP: 10.672 ms
Maximum latency for UDP: 10.672 ms
Average jitter for UDP: 1.75181e-13 ms
```

V adresáři ns-3 byl také vytvořen graf zatížení linky mezi uzly n0 a n1 případně n3 a n4. Tento graf by měl být stejný jako na Obr. 6-4.



Obr. 6-4: Graf zatížení linek pro TCP a UDP

6.1.6.1 Doplňující otázky

- Proč je zpoždění v případě TCP větší než v případě UDP?
- Proč je zatížení linky rozdílné, když jsou obě aplikace nastaveny na stejnou hodnotu 0,5 Mb/s?

6.2 Chybovost při přenosu

Nyní budeme nastavovat hodnoty chybovosti na středním uzlu každé sítě (n1 a n4). K tomu použijeme *ErrorModel* integrovaný v ns3. Pro povolení je třeba opět vložit příkazy na označená místa. Začneme nastavením parametrů modelu. Definujeme hodnotu a jednotku, podle které bude *ErrorModel* pakety zahazovat. Jednotka *ErrorRate* vyjadřuje procentuální chybovost z dlouhodobého hlediska.

```
//Nastaveni parametru Error model
Config::SetDefault("ns3::RateErrorModel::ErrorRate",DoubleValue(0));
Config::SetDefault("ns3::RateErrorModel::ErrorUnit",StringValue("ERROR_UNIT_PACKET"));
std::string errorModelType = "ns3::RateErrorModel";
```

Model následně přiřadíme k uzlům:

```
//Error model konfigurace
ObjectFactory factory;
factory.SetTypeId (errorModelType);
Ptr<ErrorModel> em = factory.Create<ErrorModel> ();
d0d1.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
d3d4.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
```

Také budeme chtít upravit zobrazované grafy tak, abychom viděli pro každou aplikaci odeslaná a přijatá data. K původnímu volání funkce *transmitPacket* volané při odeslání přidáme dvě položky, které zavolají funkci *receivedPacket* při příjmu.

```
        //Trasovani zatizeni-prijimaci strana
        Config::Connect("/NodeList/2/$ns3::Ipv4L3Protocol/Rx",MakeBoundCallback
(&receivedPacket,"tcp"));
        Config::Connect("/NodeList/5/$ns3::Ipv4L3Protocol/Rx",MakeBoundCallback
(&receivedPacket,"udp"));
```

Na prostředních uzlech budeme sledovat počet zahozených paketů. **V části //Vystup z terminalu odkomentujte položky *"Number of dropped packets tcp/udp: "*** a vložte příkazy:

```
//Trasovani zahozeni
Config::Connect("/NodeList/1/DeviceList/*/$ns3::PointToPointNetDevice/PhyRxDrop",
MakeCallback(&dropPacket));
Config::Connect("/NodeList/4/DeviceList/*/$ns3::PointToPointNetDevice/PhyRxDrop",
MakeCallback(&dropPacketUDP));
```

Pro správné zobrazení grafu zakomentujte část označenou jako //Spolecny graf a odkomentujte části //Graf TCP a //Graf UDP.

6.2.1 Zobrazení výsledků

Vytvořenou simulaci opět spusťte. A zkontrolujte výsledky, které by se neměly lišit od výsledků předchozí simulace, protože hodnotu chybovosti máme stále nastavenou na nula. Zatížení, které je viditelné v grafech, by se na straně vysílače a přijímače nemělo významně lišit.

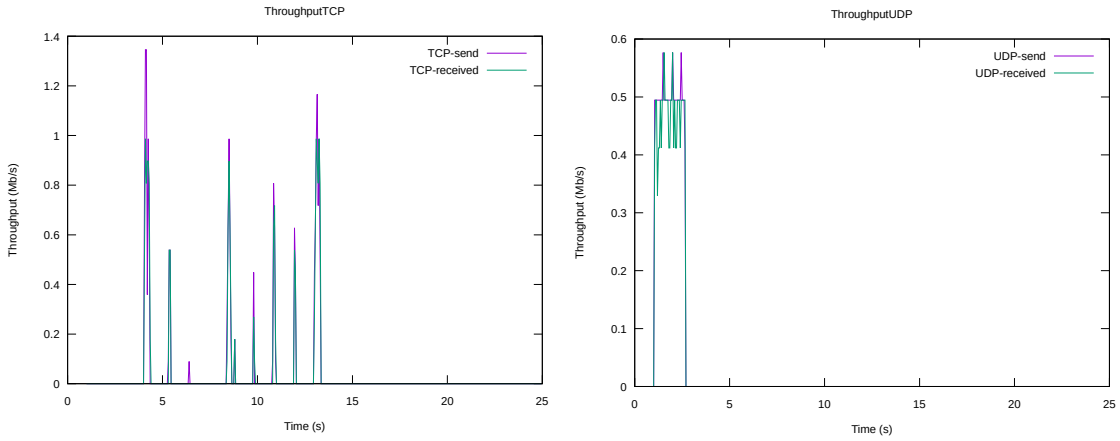
Nyní upravíme hodnotu *ErrorRate* která se nachází v části //Nastaveni parametru Error model. Hodnotu nastavíme na 0.01 což představuje 1% ztrátovosti. Před spuštěním simulátoru ještě **upravte hodnotu proměnné *step* ve funkci *saveThroughput* na 0,05.** Vzorkování tak bude probíhat desetkrát častěji a výsledky v grafu budou lépe patrné. Simulátor spusťte a sledujte změny ve zpoždění, počty zahozených paketů a chybovost. Výstup z terminálu by měl vypadat takto:

```
TCP data sent: 102400 B
UDP data sent: 102400 B
TCP data received: 102400 B, error rate: 0 %
UDP data received: 101376 B, error rate: 1 %
Average latency for TCP: 2863.99 ms
Maximum latency for TCP: 3018.29 ms
Average jitter for TCP: 4.30283 ms
Number of dropped packets tcp: 3
Number of dropped packets udp: 2
Average latency for UDP: 10.672 ms
Maximum latency for UDP: 10.672 ms
Average jitter for UDP: 1.72451e-13 ms
```

Postupně měňte hodnotu zahazování na 2, 4, 6, 10 % (0.02, 0.04, 0.06, 0.1). **Pro vyšší hodnoty chybovosti (6 a 10 %) je třeba prodloužit délku simulace a také čas vypnutí aplikací na 25 sekund**. Dále jsou pro kontrolu uvedeny výstupy z terminálu a grafy (Obr. 6-5) v případě 6 % chybovosti. Celý soubor simulace si uložte tak, aby bylo možné ho později pro kontrolu spustit, a odpovězte na doplňující otázky.

```
TCP data sent: 102400 B
UDP data sent: 102400 B
TCP data received: 102400 B, error rate: 0 %
```

```
UDP data received: 96256 B, error rate: 6 %
Average latency for TCP: 8078.6 ms
Maximum latency for TCP: 11386.8 ms
Average jitter for TCP: 60.9137 ms
Number of dropped packets tcp: 12
Number of dropped packets udp: 12
Average latency for UDP: 10.672 ms
Maximum latency for UDP: 10.672 ms
Average jitter for UDP: 1.65049e-13 ms
```



Obr. 6-5: Graf zatížení pro TCP a UDP (chybovost 6%)

6.2.2 Doplňující otázky

- Ze zachycených souborů *.pcap* zjistěte, proč v případě nastavené chybovosti začíná přenos TCP až v čase 4 sekundy, přestože má aplikace nastavené zapnutí v čase 1 sekunda.
- Jak na chyby reaguje protokol TCP a jaká byla ztrátovost jeho přenosu?
- V zachycených souborech *.pcap* nalezněte opakovaný přenos TCP. Jak je označen?
- Jak na chyby reaguje protokol UDP a jaká byla ztrátovost jeho přenosu?

6.3 Výpadek linky

Ve stejných sítích jako v případě nastavované chybovosti si ukážeme i reakci protokolů na výpadek linky kterou využívají pro komunikaci. **Hodnotu chybovosti nastavte zpět na 0 % a zakomentujte položky *"Number of dropped packets: "* v části //Vystup terminal. Délku simulace vraťte zpět na 10 sekund a na stejnou hodnotu nastavte ukončení aplikací.**

Před řádek, který spouští simulátor, vložte následující kód, kterým dojde k deaktivaci a následnému zprovoznění linky mezi n1-n2 a n4-n5. K výpadku dojde v čase 1,5 sekund od spuštění simulátoru. K obnovení dojde ve 2. sekundě.

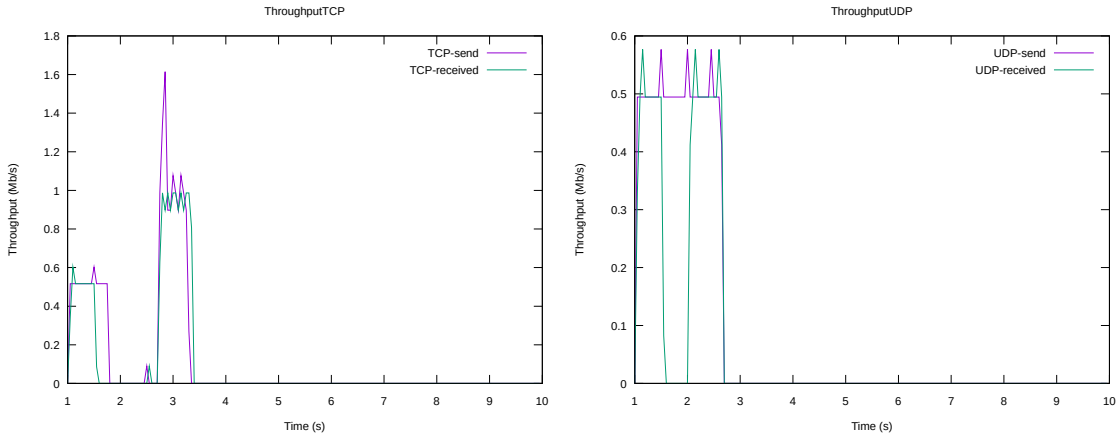
```
//Vypadek
Ptr<Ipv4> IPv4 = n.Get(1)->GetObject<Ipv4> ();
       Simulator::Schedule(Seconds(1.5), &Ipv4::SetDown, IPv4, 1);
       Simulator::Schedule(Seconds(2.0), &Ipv4::SetUp, IPv4, 1);
Ptr<Ipv4> IpV4 = n.Get(4)->GetObject<Ipv4> ();
       Simulator::Schedule(Seconds(1.5), &Ipv4::SetDown, IpV4, 1);
       Simulator::Schedule(Seconds(2.0), &Ipv4::SetUp, IpV4, 1);
```

6.3.1 Zobrazení výsledků

Simulátor nyní opět spusťte. Výstup z terminálu by měl vypadat takto:

```
TCP data sent: 102400 B
UDP data sent: 102400 B
TCP data received: 102400 B, error rate: 0 %
UDP data received: 71168 B, error rate: 30.5 %
Average latency for TCP: 681.743 ms
Maximum latency for TCP: 1205.09 ms
Average jitter for TCP: 8.57649 ms
Average latency for UDP: 8.672 ms
Maximum latency for UDP: 8.672 ms
Average jitter for UDP: 1.36767e-13 ms
```

Prohlédněte si také vytvořené grafy (Obr. 6-6) a odpovězte na následující doplňující otázky. Kód k simulaci si opět ponechejte uložen pro pozdější kontrolu.



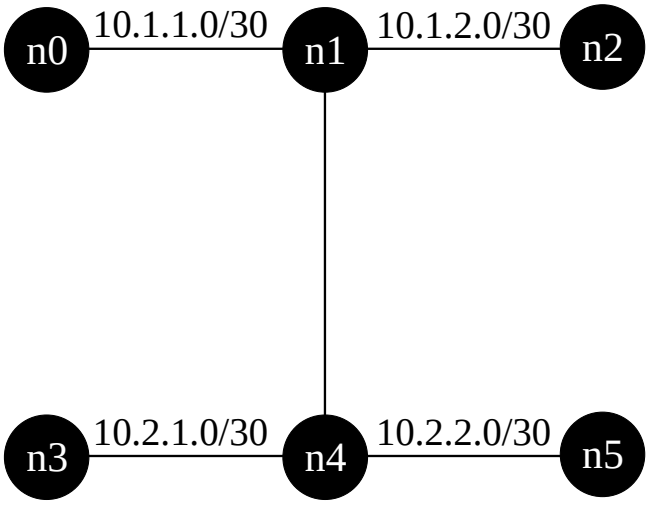
Obr. 6-6: Graf zatížení při výpadku pro TCP a UDP

6.3.2 Doplňující otázky

- Jak protokoly reagují na výpadek?
- Jak je možné že TCP využívá celou šířku pásma i když má nastavenou rychlost 0,5Mb/s?
- Z grafu Zatížení TCP je patrný menší nárůst v čase 2,5 sekund. Ze souborů *.pcap* zjistěte, o jaký provoz se jedná.

6.4 Přetížení linky

V poslední části vytvoříme podle Obr. 6-7 společnou síť pro obě aplikace. TCP bude zasílat data z uzlu n0 na uzel n3. UDP z uzlu n2 na uzel n5.



Obr. 6-7: Společná topologie pro TCP i UDP

Vycházet budeme z předchozí simulace tak, že doplníme linku mezi uzly n1 a n4. Tím dojde k propojení samostatných sítí používaných v předchozích úlohách. **Nejprve je však nutné odstranit kód, kterým jsme v předchozím kroku nastavovali výpadek.** Následně vytvoříme nový kontejner pro spojení uzlů. Kód vložte na místo, kde se nachází původní definice uzlů.

```
NodeContainer n1n4 = NodeContainer (n.Get (1), n.Get (4));
```

Ke kontejneru doplníme spojení typu point-to-point. Následující kód vložte na místo, kde jsou vytvořeny původní linky. **A samostatně přiřaďte vhodný adresní rozsah podle vlastního uvážení.**

```
NetDeviceContainer d1d4 = p2p.Install (n1n4);
```

Dále je třeba upravit parametry simulace. **Rychlost linek nastavíme na 0.8 Mb/s, délku simulace na 20 sekund a proměnnou *step* ve funkci *saveThroughput* vrátíme zpět na 0.5. Aplikace nastavte podle Tab. 6-1.**

Tab. 6-1: Nastavení parametrů aplikací

	Komunikace	Velikost odeslaných dat	Zapnutí	Vypnutí
TCP	z n0 na n3 (i3i4.GetAddress (0))	1024000 B (proměnná MaxBytes)	1 s	20 s
UDP	z n2 na n5 (i4i5.GetAddress (1))	Neomezeno **(řádek který hodnotu nastavuje zakomentujte – onoffUDP.SetAtribute())**	6 s	10 s

Posledním krokem je úprava trasování, protože došlo ke změně uzlů, na kterých běží aplikace. Volání funkce *transmitPacket* ponecháme na uzlu n0 pro TCP. Změnit je třeba volání pro UDP, a to na uzel n2 takto:

```
Config::Connect("/NodeList/0/$ns3::Ipv4L3Protocol/Tx",MakeBoundCallback
(&trasmitPacket, "tcp"));
```

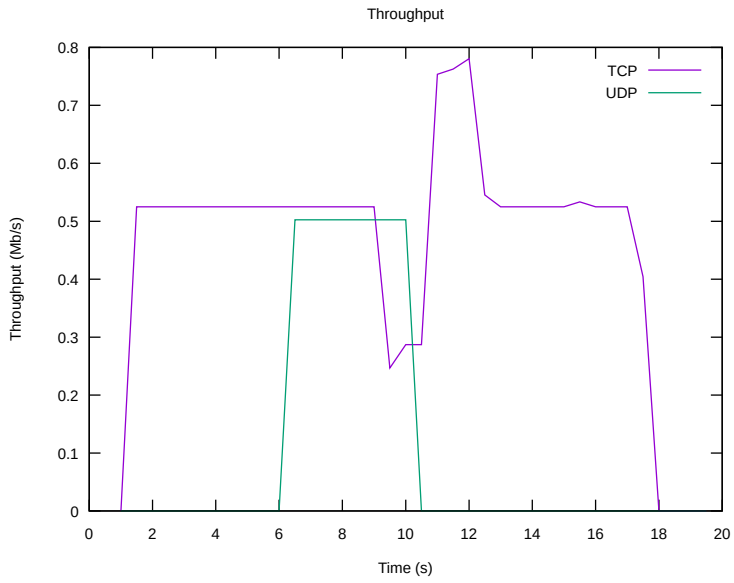
```
        Config::Connect("/NodeList/2/$ns3::Ipv4L3Protocol/Tx",MakeBoundCallback
(&trasmitPacket, "udp"));
```

Samostatně změňte uzly, na kterých jsou volány funkce *sendPacket* a *receivePacket*. Volání funkcí *receivedPacket* (//Trasovani zatizeni-prijimaci strana) můžeme zakomentovat. V této části je nebudeme používat.

V grafu budeme chtít vidět zatížení linek protokolem TCP a UDP v jednom grafu. Zakomentujte tedy části //Graf TCP, //Graf UDP a odkomentujte část //Spolecny graf.

6.4.1 Zobrazení výsledků

Simulátor spustíme a zobrazíme si uložený graf, který je také zobrazen na Obr. 6-8. Odpovězte na doplňující otázky.



Obr. 6-8: Zatížení při přetížení linky

6.4.2 Doplňující otázky

- Jak zareaguje protokol TCP na přítomnost dalšího provozu na lince? Jaké mechanismy k tomu používá a jak fungují?
- Proč k přetížení linky došlo?
- Samostatně si změňte přenosovou rychlost linek na hodnotu větší než 1 Mb/s. Graf opět okomentujte.

6.5 Protokol SCTP

V této části si předvedeme funkci protokolu SCTP. SCTP není v ns-3 přímo podporován a je tak nutné využít implementací linuxového jádra pomocí ns-3 DCE. Bohužel však není možné použít trasování pro zobrazení grafů a podobně.

Přepněte se do virtuálního prostředí, kde je DCE nainstalován. Pro předvedení funkce budeme využívat příkladů, které jsou součástí ns-3, později je mírně upravíme. Přejděte do adresáře ~/dce/source/ns-3-dce/example a otevřete soubor *dce-sctp-simple.cc*. Všimněte si, kolik uzlů je vytvořeno a jak jsou propojeny. Odpovězte na otázky:

- Jaké IP adresy mají uzly přiděleny?
- Který uzel figuruje jako klient a který jako server?

Nyní si uvedenou simulaci spustíme. V terminálu přejdeme do adresáře ~/dce/source/ns-3-dce a spustíme příkazem **./waf --run dce-sctp-simple**. V adresáři ns-3-dce se nám vytvoří dva zachycené soubory *.pcap*. Soubory zachycené na uzlu 0 by po otevření měly odpovídat následujícímu Obr. 6-9Obr. 6-8.

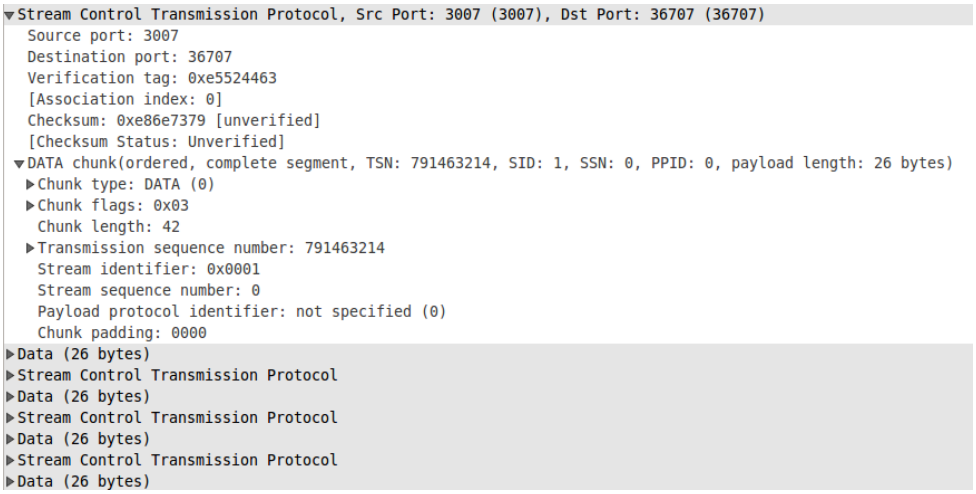
3	1.502000	10.0.0.2	10.0.0.1	SCTP	70	INIT
4	1.502000	10.0.0.1	10.0.0.2	SCTP	294	INIT_ACK
5	1.506001	10.0.0.2	10.0.0.1	SCTP	266	COOKIE_ECHO
6	1.506001	10.0.0.1	10.0.0.2	SCTP	38	COOKIE_ACK
7	1.506001	10.0.0.1	10.0.0.2	SCTP	78	DATA
8	1.506001	10.0.0.1	10.0.0.2	SCTP	1486	DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA
9	1.506003	10.0.0.1	10.0.0.2	SCTP	1486	DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA
10	1.506005	10.0.0.1	10.0.0.2	SCTP	1486	DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA
11	1.506008	10.0.0.1	10.0.0.2	SCTP	1486	DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA
12	1.506010	10.0.0.1	10.0.0.2	SCTP	1486	DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA
13	1.510001	10.0.0.2	10.0.0.1	SCTP	50	SACK
14	1.510006	10.0.0.2	10.0.0.1	SCTP	50	SACK
15	1.510008	10.0.0.2	10.0.0.1	SCTP	42	SHUTDOWN
16	1.510008	10.0.0.1	10.0.0.2	SCTP	1486	DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA DATA
17	1.510010	10.0.0.1	10.0.0.2	SCTP	78	DATA
18	1.510010	10.0.0.2	10.0.0.1	SCTP	58	SHUTDOWN SACK
19	1.510013	10.0.0.2	10.0.0.1	SCTP	58	SHUTDOWN SACK
20	1.514010	10.0.0.2	10.0.0.1	SCTP	58	SHUTDOWN SACK
21	1.514010	10.0.0.2	10.0.0.1	SCTP	58	SHUTDOWN SACK
22	1.514010	10.0.0.1	10.0.0.2	SCTP	38	SHUTDOWN_ACK
23	1.518011	10.0.0.2	10.0.0.1	SCTP	38	SHUTDOWN_COMPLETE

Obr. 6-9: Zachycená komunikace pomocí protokolu SCTP

V první řadě se zaměřte na navazování a ukončování spojení.

- Jaká čísla jsou při navazování vyměňována?
- Jak se spojení nazývá?

Dále si otevřete paket přenášející data. Ten by měl vypadat jako na Obr. 6-10.

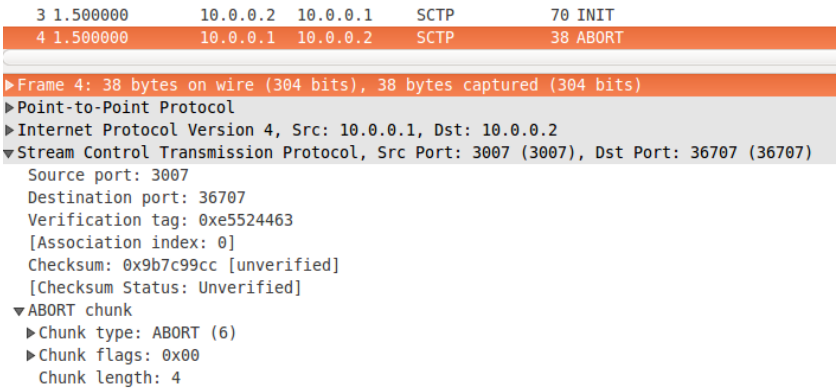


Obr. 6-10: Zachycený datový blok (data chunk)

Popište, co je přenášeno v hlavičce datového bloku (data chunk). Klient je nastaven tak, aby od serveru přijal celkem 100 datových bloků, naopak server jich odesílá 200. Toto chování zkuste v zachyceném souboru nalézt a z informací uvedených v teoretickém úvodu vlastními slovy popište, k čemu dochází. Odpovězte také na otázku

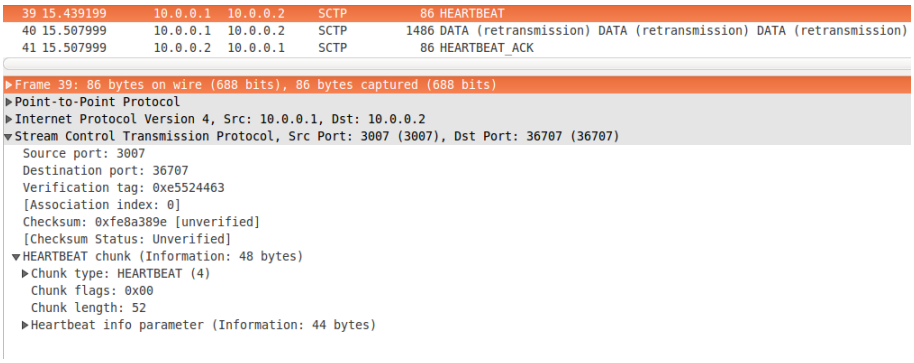
- Jak probíhá potvrzování?

V souboru *dce-sctp-simple.cc* změňte čas spuštění serveru na 5 sekund. Simulaci spusťte a otevřete nový soubor *.pcap* (Obr. 6-11). Popište, k čemu touto změnou došlo a jak na to zareagoval protokol.



Obr. 6-11: Ukončení při navazování asociace

Vraťte hodnotu spuštění serveru zpět na 1 sekundu a změňte hodnotu rychlosti linky (*DataRate*) na 10 kbps. Tím dojde k opakovaným přenosům. Tyto opakované přenosy najděte a popište, jak jsou označeny. V zachyceném souboru také nalezněte zprávu *HEARTBEAT* (Obr. 6-12), k čemu zpráva slouží a jak je potvrzována?



Obr. 6-12: SCTP HEARTBEAT blok (chunk)

6.5.1 Doplňující otázky

- V kolika samostatných proudech jsou data v rámci simulace přenášena?
- K čemu slouží datový blok (*chunk*) a kolik takových bloků je možné v rámci jednoho paketu přenášet?
- Jak je možné že server zasílal data i po přijetí zprávy *SHUTDOWN?*
- Kterému ze základních transportních protokolů (TCP, UDP) je protokol SCTP podobnější?
- Kde vidíte možné použití protokolu?

7. SÍŤOVÉ PRVKY, TOPOLOGIE, SMĚROVÁNÍ

Tato kapitola slouží jako teoretický úvod k laboratorní úloze, která se zabývá rozdíly mezi zařízeními používanými v síti (přepínač, směrovač). Je zde také popsán protokol ARP, který slouží k nalezení fyzické adresy. V případě směrování je popsán směrovací protokol RIPv2. Probrány jsou také možnosti síťových topologií.

7.1 Přepínač

Přepínač neboli switch je zařízení s větším množstvím portů, které pracuje na linkové vrstvě. Podle MAC adresy, která je uvedena v příchozím rámci, se přepínač rozhoduje, na který port rámec odešle. Běžný provoz je tedy odesílán pouze na jeden port což snižuje zatížení sítě, které vzniká při použití hubů. Tabulku portů a MAC adres si přepínač udržuje automaticky bez nutnosti zásahu. Vytváří ji z provozu, který na switch přichází.

Přepínače nahradily dříve používané huby a nejčastěji se používají v ethernetových sítích. Kromě nižšího zatížení má switch pozitivní vliv na bezpečnost v síti, protože se data vysílají pouze do rozhraní, na kterém je připojen příjemce.

Často se můžeme setkat s L3 přepínači, které pracují na třetí vrstvě. Právě kvůli funkci na síťové vrstvě se ale spíše jedná o směrovač. Ten je popsán v další kapitole. Funkci přepínače si vysvětlíme na následujícím příkladu.

Představme si síť s několika koncovými stanicemi připojenými k jednomu přepínači. Stanice mají přiděleny IP adresy. To jim ale pro komunikaci mezi sebou nepomůže, protože potřebují znát fyzickou MAC adresu stanice, se kterou chtějí komunikovat. Ke zjištění MAC adresy slouží například dynamický protokol ARP, který je popsán v kapitole 7.3. Je možné tabulky vytvářet i staticky což ale přináší problémy s jejich správou. První rámec přepošle přepínač všesměrově, protože neví, ke kterému portu příjemce náleží. Pomocí metody zpětného učení (backward learning) si přepínač na základě příchozích rámců vytváří přepínací tabulku.

7.2 Směrovač

Jak již bylo řečeno, směrovač pracuje na třetí (síťové) vrstvě. A slouží především k propojování sítí. Směrovač podle cílové adresy umístěné v hlavičce paketu předává provoz k příjemci. Každé rozhraní, které směrovač obsahuje (typicky dvě a více), musí mít přidělenou vlastní IP adresu.

Hlavní úlohou směrovače je směrování. K tomu jsou používány směrovací protokoly. Ty směrovači zajišťují informace o sousedech a z nich tvoří směrovací tabulky. Mezi směrovací protokoly používané v autonomních systémech (souhrn sítí pod společnou správou [17]) řadíme například RIP (Routing Information Protocol) či OSPF (Open Shortest Path First). Popis protokolu RIPv2 je uveden v další kapitole. Mezi autonomními systémy je zpravidla využíván protokol BGP (Border Gateway Protocol).

7.3 Protokol ARP

Protokol ARP (Address Resolution Protocol) slouží nejčastěji k nalezení fyzické adresy k IP adrese. Protokol funguje dynamicky a je schopen reagovat na změny v síti. Informace o odpovídajících adresách si protokol ukládá do tabulky. Záznamy v tabulkách jsou zpravidla platné po omezenou dobu a pak z tabulky odstraněny, tak, aby nedošlo k neaktuálnosti. Struktura ARP paketu je zobrazena na Obr. 7-1. V následujícím textu je uvedeno vysvětlení jednotlivých polí.

Obr. 7-1: Struktura ARP paketu

- **Typ média** (Hardware type) – 16bitové pole, které definuje typ sítě, na které ARP běží. Například ethernetu bylo přiděleno číslo 1 [16].

- **Typ protokolu** (Protocol type) – další pole o délce 16 bitů. Definuje typ protokolu vyšší vrstvy. Hodnota pole pro IPv4 je 0800_{16} [16].
- **Délka fyzické adresy** (Hardware length) – 8 bitů. Délka fyzické adresy. V případě ethernetu je délka 6.
- **Délka logické adresy** (Protocol length) – 8 bitů. Délka logické adresy. V případě IPv4 je délka 4.
- **Operace** (Operation) – 16 bitů, které definují typ zasílaného paketu. ARP definuje pouze dva typy. Typ 1 – pro požadavek (ARP request) a Typ 2 – slouží k odpovědi (ARP reply).
- **Fyzická adresa zdroje** (Sender hardware address) – Pole proměnlivé délky, ta je specifikována v poli *délka fyzické adresy*. Definuje fyzickou adresu zdroje. Typicky MAC adresa.
- **Logická adresa zdroje** (Sender logical address) - Pole proměnlivé délky, ta je specifikována v poli *délka logické adresy*. Definuje logickou adresu zdroje. Typicky IP adresa.
- **Hledaná fyzická adresa** (Target hardware address) - Pole proměnlivé délky, ta je specifikována v poli *délka fyzické adresy*. Definuje fyzickou adresu, která je hledána. Typicky MAC adresa.
- **Hledaná logická adresa** (Target logical address) - Pole proměnlivé délky, ta je specifikována v poli *délka logické adresy*. Definuje logickou adresu, která je hledána. Typicky IP adresa.

Předpokládejme, že stanice A chce komunikovat se stanicí B, obě se nachází ve stejné síti, a mohou tak spolu přímo komunikovat. Síťová vrstva získala od transportní vrstvy data s IP adresou stanice B. K vytvoření rámce musí být stanice A schopna převést IP adresu na fyzickou adresu. Stanice A tedy prozkoumá ARP tabulku se záznamy, a pokud nenajde vhodný záznam, použije protokol ARP k položení dotazu. V dotazu, který vyšle všesměrově do sítě, jsou uvedeny IP adresa a fyzická adresa stanice A a hledaná IP adresa stanice B. Stanice B odpoví ARP zprávou, která je zaslána přímo na stanici A pomocí fyzické adresy. Ta nese i vyplněnou fyzickou adresu stanice B. Ostatními stanicemi je všesměrový požadavek zahozen. V případě že se komunikující uzly nachází

v jiné síti, odesílají se rámce na fyzickou adresu výchozí brány. Pokud je adresa brány neznámá, probíhá zjištění pomocí ARP stejně jako v předchozím případě.

7.4 RIPv2

Protokol RIP řadíme ke směrovacím protokolům typu *Distance vector*, kde se o výběru nejlepší cesty rozhoduje na základě počtu skoků. Jedná se o protokol, který je schopen reagovat na změny v síti a podle nich upravit směrovací tabulku. Ta je vytvářena automaticky na základě informací od ostatních směrovačů. Každá tabulka tak obsahuje adresu sítě, rozhraní, které k síti vede, a počet skoků, což je metrika protokolu RIP.

Aktualizační zprávy (*update message*) si směrovače vyměňují pravidelně, případně dojde-li ke změně v síti. Podle informací, které jsou ve zprávě obsaženy, je upravena směrovací tabulka tak, že je zapsána nová cesta, a k hodnotě metriky ze zprávy je přičten jeden skok. Směrovač, který zprávu update odeslal je označen jako nejbližší skok (*next hop*). Po aktualizaci vlastní tabulky vysílá směrovač zprávu update s novými informacemi [10].

Smyčkám protokol předchází omezením maximálního počtu skoků na 15. Každá zpráva update, která změní metriku na 16, je pro RIP nekonečně vzdálena a tím nedostupná. Tím je ale omezena maximální velikost sítě [21].

7.4.1.1 Formát zprávy RIPv2

Formát paketu protokolu RIPv2 je na Obr. 7-2. Dále jsou popsána jednotlivá pole.

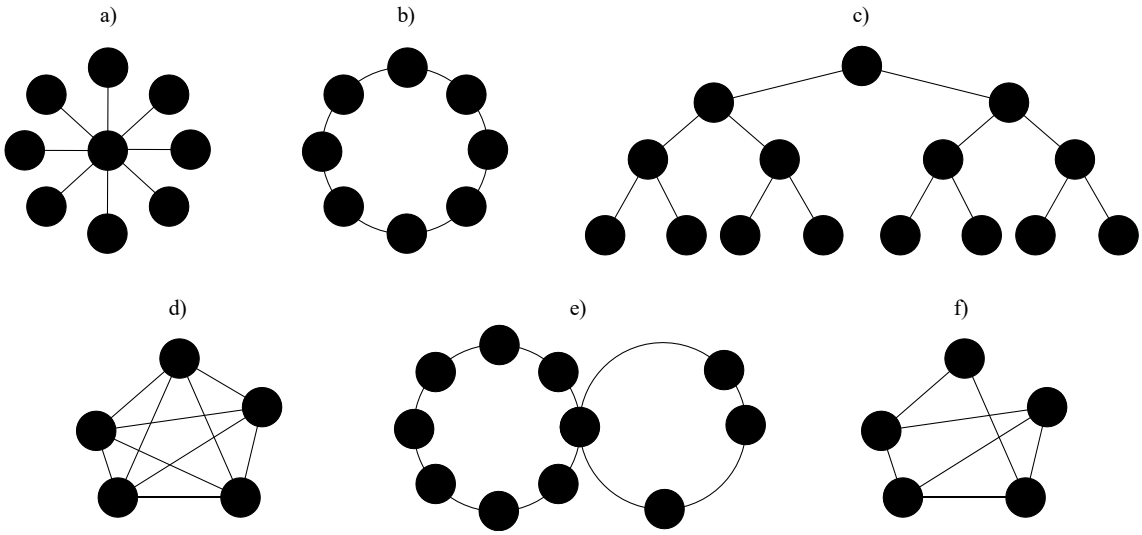
Obr. 7-2: Struktura paketu RIPv2 [21]

- **Command** – 8 bitů, které nesou informaci o tom, zda jde o paket nesoucí požadavek nebo odpověď. Odpovědi nesou informace vedoucí k aktualizaci směrovací tabulky [10] [21].
- **Číslo verze** (Version number) – 8 bitů, sloužící ke specifikaci verze RIP. V případě RIPv2 je hodnota nastavena na 2 [21].
- **Rezerva** (Reserved) – 8 bitů.
- **Identifikátor adresní rodiny** (AFI) – 16 bitů. Definuje použitou adresní rodinu. Například pro IP je hodnota nastavena na 2. V případě, že má pole prvního záznamu hodnotu 0xFFFF, je použita autentifikace a zbytek záznamu nese informace o ní [10] [21].
- **Směrovací značka** (Route tag) – pole o velikosti 16 bitů. Slouží k rozlišení, zda jde o směrování naučené protokolem RIP (interním), případně od ostatních protokolů (externích) [21].
- **Síťová adresa** (Network address) – z logiky věci 32 bitů. Specifikuje IP adresu záznamu [21].
- **Maska podsítě** (Subnet mask) – opět 32 bitů, které nesou masku podsítě. RIP tedy podporuje beztřídní adresování [21].
- **Další skok** (Next hop) – Specifikuje adresu dalšího skoku, na kterou jsou pakety přeposílány- 32 bitů [21].
- **Metrika** (Metric) – Definuje počet skoků od směrovače, který cestu nabízí do cílové sítě. Hodnota mezi 1 a 15. 16 se považuje za nekonečno [21].

7.5 Topologie sítí

Později v laboratorní úloze budeme vytvářet základní topologie, skládající se ze spojení dvou bodů [17]. Základní přehled je uveden v následujícím textu, včetně grafické ukázky na Obr. 7.3.

- Hvězda (a)
- Kruh (b)
- Strom (c)
- Úplný polygon (propojení každý s každým) (d)
- Propojené kruhy (e)
- Obecná topologie (neúplný polygon) (f)



Obr. 7-3: Přehled základních topologií [17]

8. ÚLOHA 3: SROVNÁNÍ SÍŤOVÁCH PRVKŮ, TOPOLOGIÍ A SMĚROVÁNÍ V NS-3

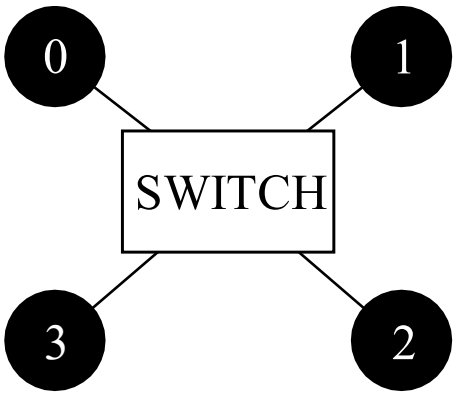
Laboratorní úloha je zaměřená na tvorbu jednoduchých síťových topologií. Vytvoříme síť zapojenou do hvězdy, obecnou topologii a stromovou strukturu sítě. V simulacích si ukážeme rozdíl mezi přepínačem a směrovačem. V případě směrovačů budeme využívat globálního směrování Ns-3, které vychází z protokolu OSPF. Blíže se pak podíváme na protokol RIPv2

8.1 Tvorba modelu – hvězdicová topologie

V první části úlohy vytvoříme jednoduchou hvězdicovou topologii. Nejprve použijeme jako centrální prvek jednoduchý přepínač, později ho nahradíme za směrovač a budeme sledovat změny.

8.1.1 Přepínač

Základní topologie pro simulaci s přepínačem, budeme zapojovat podle Obr. 8-1.



Obr. 8-1: Topologie s přepínačem

K vytvoření budeme potřebovat čtyři uzly a jeden přepínač. Pro spojení budeme využívat technologii s přístupovou metodou CSMA/CD (Carrier Sense Multiple Access / Collision Detection), tedy metodu používanou technologií Ethernet. K testování funkčnosti zapojení nám bude sloužit jednoduchá UDP aplikace.

Otevřete si předpřipravený soubor *switch_vychozi.cc*, který budeme postupně upravovat vkládáním kódu na označená místa. K editaci můžete použít jakýkoliv textový editor případně prostředí *Eclipse*. Nejprve vytvoříme dva kontejnery uzlů. Jeden pro uzly

označené na Obr. 8-1 jako n0-n3 a druhý pro samotný přepínač. To uděláme vložením následujícího kódu.

```
//Tvorba uzlu
NodeContainer terminals;
terminals.Create (4);

NodeContainer csmaSwitch;
csmaSwitch.Create (1);
```

Pro definici linky, která bude propojovat uzly s přepínačem, využijeme *CsmaHelper*, pomocí něhož vytvoříme CSMA spojení, a tomu také přiřadíme parametry. Rychlost bude nastavena na 10Mb/s a zpoždění nastavíme na 0ms, protože nás bude zajímat hlavně zpoždění na zařízeních. To vše uděláme vložením následujícího kódu.

```
//Definice linky
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("10Mb/s"));
csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (0)));
```

Následně vytvoříme rozhraní uzlů a přepínače pomocí *NetDeviceContainer*. Vytváření probíhá, podobně jako v případě uzlů, následovně.

```
//Tvorba rozhrani
NetDeviceContainer terminalDevices;
NetDeviceContainer switchDevices;
```

Rozhraní propojíme s přepínačem CSMA linkou. Abychom nemuseli každou linku vytvářet zvlášť, použijeme následující cyklus.

```
//Propojeni - cyklus
for (int i = 0; i < 4; i++) {
  NetDeviceContainer link=csma.Install (NodeContainer (terminals.Get (i), csmaSwitch));
  terminalDevices.Add (link.Get (0));
  switchDevices.Add (link.Get (1));}
```

Cyklus si nejdříve projděte a snažte se pochopit, jak funguje. Co představuje proměnná i? K čemu v tomto případě slouží *NodeContainer*? Všimněte si také, jak probíhá v ns-3 číslování. Od jakého čísla je číslováno?

Přepínač je v tuto chvíli stále jen jednoduchý uzel s vazbou na terminály. Pro funkčnost mu musíme přiřadit funkci přepínače. K tomu využijeme *BridgeHelper*, čímž vytvoříme jednoduchý přepínač, a přiřadíme ho našemu uzlu.

```
//Tvorba prepinace
Ptr<Node> switchNode = csmaSwitch.Get (0);
BridgeHelper swtch;
swtch.Install (switchNode, switchDevices);
```

Před tím, než provedeme přiřazení IP adres k terminálům, musíme na ně ještě nainstalovat protokolový zásobník.

```
//Instalace protokoloveho zasobniku
InternetStackHelper internet;
internet.Install (terminals);
```

Nyní můžeme přejít k přiřazení adres. Ty jsou na terminály přidělovány postupně podle pořadí uzlů. K přiřazení adres nám pomůže *Ipv4AddresHelper*.

```
//Prirazeni IP adres
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
ipv4.Assign (terminalDevices);
```

Tím jsme dokončili kompletní konfiguraci naší sítě. Nyní si vygenerujeme provoz na ní. Použijeme k tomu jednoduchou UDP aplikaci, která bude zasílat datagramy z terminálu 0 (přidělená adresa 10.1.1.1) na terminál 1 (přidělená adresa 10.1.1.2). Celkově zašle 10 datagramů které přenesou 5120 bajtů. Aplikaci vytvoříme zadáním následujícího kódu.

```
//UDP Aplikace
uint16_t port = 9;   // Discard port (RFC 863)
uint32_t maxBytes = 512*10;
```

```
        //Terminal 0 - odesilani
        OnOffHelper  onoff  ("ns3::UdpSocketFactory",Address(InetSocketAddress
("10.1.1.2", port)));
        onoff.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
        onoff.SetConstantRate (DataRate ("1Mb/s"));
        ApplicationContainer apps = onoff.Install (terminals.Get (0));
        apps.Start (Seconds (1.0));
        apps.Stop (Seconds (9.0));
        //Terminal 1 - prijem
        PacketSinkHelper sink("ns3::UdpSocketFactory",Address(InetSocketAddress
(Ipv4Address::GetAny (), port)));
        apps = sink.Install (terminals.Get (1));
        apps.Start (Seconds (0.0));
```

Pro zobrazení výsledků budeme používat soubory zachycené na každém rozhraní každého uzlu. Jedná se o soubory *.pcap*, které je možné otevřít v programu *Wireshark*. Zachycení souboru nastavíme tímto kódem.

```
        //Zachytavani .pcap
        csma.EnablePcapAll ("switch", false);
```

Kromě zachycených souboru si vytvoříme grafy zpoždění pro každý z datagramů. K tomu je určena třída database a několik dalších metod, které jsou volány v případě odeslání a doručení datagramu. Třída i metody jsou umístěny v samostatném souboru *Database_topo*. Kód, který je uvnitř, zkopírujte před hlavní funkci *main*.

Nyní máme vytvořeny metody a databáze, které je nutné naplnit daty. V případě odeslání a přijetí datagramu bude zavolána funkce *sendPacket a receivedPaket*. K tomu využijeme trasování ns-3, které povolíme následujícím kódem.

```
//Trasovi - aplikace
Config::Connect("/NodeList/0/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "udp"));
Config::Connect("/NodeList/1/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "udp"));
```

Další funkcí, kterou zavoláme, je funkce *saveLatency*, ke každému datagramu uloží hodnotu zpoždění. Tyto hodnoty využijeme k vykreslení grafu zpoždění.

```
//Volani saveLatency
Simulator::Schedule(Seconds(10), &saveLatency);
```

Poslední část kódu, kterou je třeba vložit, jsou řádky, které slouží ke spuštění simulátoru. Simulace je ukončena v čase 10 sekund.

```
//Spusteni simulatoru
Simulator::Stop (Seconds (10));
Simulator::Run ();
Simulator::Destroy ();
```

Na konci výchozího souboru je zakomentováno několik položek. První část nám do terminálu vypíše hodnoty přijatých a odeslaných dat, chybovost, průměrné zpoždění a zpoždění prvního a pátého datagramu. Druhá část slouží k vykreslení grafu zpoždění na pořadovém čísle datagramu.

8.1.1.1 Spuštění simulátoru a zobrazení výsledků

Soubor, který jsme postupně vytvořili, nyní zkopírujeme do adresáře ~/ns-allinone-3.27/ns-3.27/scratch. Otevřeme terminál a přejdeme do adresáře ~/ns-allinone-3.27/ns-3.27. Simulátor spustíme zadáním příkazu ./waf --run switch_vychozi. Pokud vše proběhlo v pořádku, měl by se v terminálu vypsat následující výpis.

```
UDP data sent: 5120 B
UDP data received: 5120 B, error rate: 0 %
Average latency for UDP: 1.01348 ms
Latency first datagram: 2.0996 ms
Latency fifth datagram: 0.8928 ms
```

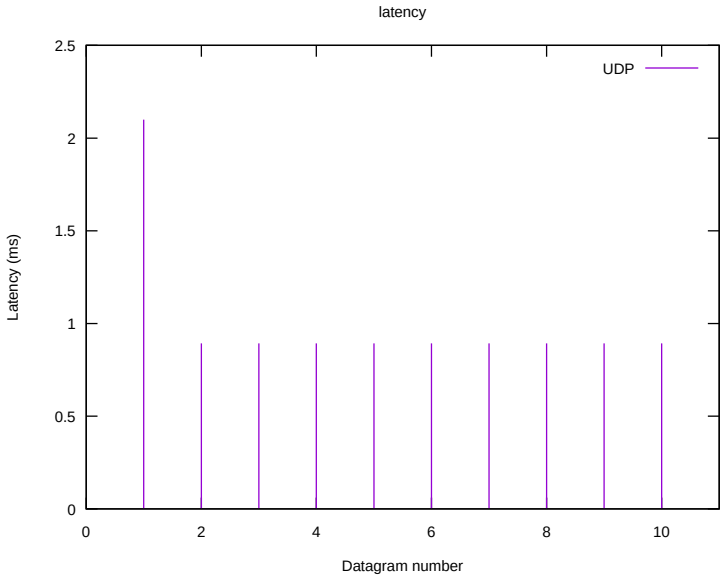
Všechny výstupní soubory jsou uloženy v adresáři, ze kterého se simulátor spouští. Jedná se o soubory *.pcap*, které jsou pojmenovány ve formátu *switch-cislo_uzlu-cislo_rozhrani.pcap*. Přepínač je v našem případě označen číslem 4. Otevřete soubor zachycený na terminálu 0, ze kterého jsou zasílána data. Měl by vypadat jako na Obr. 8-2. Komunikaci si prohlédněte a odpovězte na otázky.

No.	Time	Source	Destination	Protocol	Length	Info
1	16:00:01.005095	00:00:00_00:00:01	Broadcast	ARP	64	Who has 10.1.1.2? Tell 10.1.1.1
2	16:00:01.005301	00:00:00_00:00:03	00:00:00_00:00:01	ARP	64	10.1.1.2 is at 00:00:00:00:00:03
3	16:00:01.005301	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
4	16:00:01.008191	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
5	16:00:01.012287	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
6	16:00:01.016383	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
7	16:00:01.020479	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
8	16:00:01.024575	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
9	16:00:01.028671	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
10	16:00:01.032767	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
11	16:00:01.036863	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512
12	16:00:01.040959	10.1.1.1	10.1.1.2	UDP	558	49153 → 9 Len=512

Obr. 8-2: Komunikace zachycená na terminálu 0

- K čemu slouží první dva zachycené pakety a jaký je jejich obsah?
- Na které porty přepínače jsou ARP pakety rozesílány?

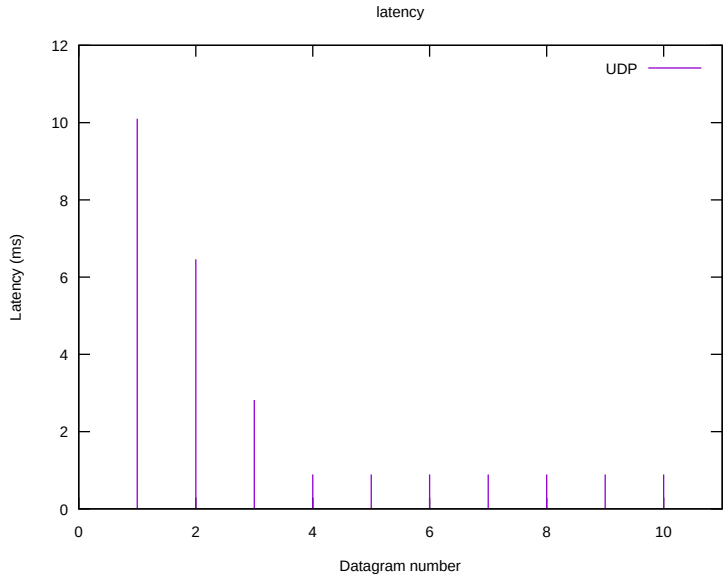
Dále si otevřete vytvořený graf, který je uložen pod názvem *latency.png* (Obr. 8-3).



Obr. 8-3: Zpoždění datagramu - 4 připojené terminály

Všimněte si většího zpoždění prvního datagramu, to je patrné i ve výstupu z terminálu. Samostatně změňte počet připojených terminálů na 6 a 8 (Obr. 8-4). Nezapomeňte, že je třeba upravit i hodnoty v cyklu for. Sledujte změnu zpoždění a odpovězte na otázky.

- K čemu slouží protokol ARP?



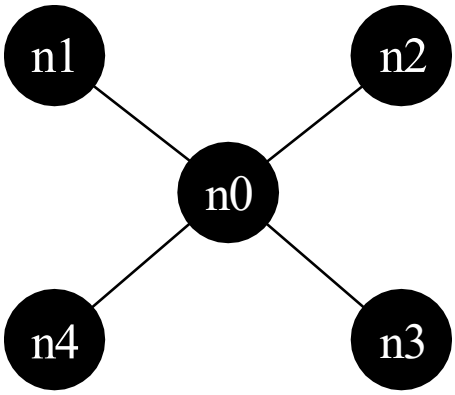
Obr. 8-4: Zpoždění datagramu - 8 připojených terminálů

- Proč je na začátku přenosu větší zpoždění a proč se s počtem terminálů zvětšuje?
- Jaké adresy přepínač používá ke správnému doručení?
- Na jaké vrstvě síťového modelu přepínač pracuje?

Zdrojový kód si uložte pro případ kontroly a přejděte k další části úlohy.

8.1.2 Směrovač

Abychom měli tvorbu topologie, která je na Obr. 8-5, jednodušší, využijeme třídu *PointToPointStarHelper*, která vytvoří veškeré spojení za nás. V této části budeme používat jednoduchá spojení bod-bod. Kód vkládejte do předpřipraveného souboru *router_vychozi.cc*.



Obr. 8-5: Topologie se směrovačem

Nejprve je třeba definovat počet uzlů připojených k centrálnímu směrovači. Počet uzlů definujeme jednoduchou proměnnou.

```
//Pocet uzlu
uint32_t n = 4;
```

Celkově bude vytvořeno n+1 uzlů. Centrální uzel bude uzel číslo 0. Linku *point-to-point*, kterou budeme uzly propojovat, definujeme následujícím kódem. Ten je podobný jako v případě linky *csma*, kterou jsme používali v předchozí úloze.

```
//Definice linky
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mb/s"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("0ms"));
```

V dalším kroku využijeme již zmíněný *PointToPointStarHelper*, který provede propojení za nás. Stačí k tomu tento jednoduchý příkaz.

```
//Star topology
PointToPointStarHelper star (n, pointToPoint);
```

Tímto máme vytvořenou hvězdu s centrálním směrovačem a k němu připojeny čtyři uzly. Na všechny teď musíme nainstalovat protokolový zásobník.

```
//Protokolovy zasobnik
InternetStackHelper internet;
star.InstallStack (internet);
```

Přiřazení IP adres provedeme známým příkazem takto.

```
//Prirazeni IP adres
star.AssignIpv4Addresses(Ipv4AddressHelper ("10.0.0.0","255.255.255.0"));
```

Protože se nyní zabýváme směrovačem, je třeba použít směrovací protokol. To nám v ns-3 zajistí *Ipv4GlobalRoutingHelper*. Globální směrování je v podstatě obdoba směrovacího protokolu OSPF bez signalizace. Všechny uzly v síti se tak budou chovat jako OSPF směrovače a budou moci komunikovat. Globální směrování povolíme následujícím příkazem.

```
//Povoleni smerovani
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Ke generování provozu budeme používat stejnou UDP aplikaci jako v předchozí úloze. Provoz bude probíhat z uzlu 1 na uzel 3. Jako ukazatel na jednotlivé uzly nám slouží funkce *GetSpokeNode*, která nám vrátí uzel připojený k terminálu. Uzly jsou číslovány od 0. Pokud tedy využijeme *star.GetSpokeNode(1)*, jedná se v celkovém pořadí o uzel 2 (tak budou označeny například soubory *.pcap)*.

```
//UDP aplikace
uint16_t port = 9;   // Discard port (RFC 863)
uint32_t maxBytes = 512*10;
//Uzel 1 - odesilani
OnOffHelper onoff ("ns3::UdpSocketFactory",Address (InetSocketAddress
(star.GetSpokeIpv4Address(3), port)));
onoff.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
onoff.SetConstantRate (DataRate ("1Mb/s"));
```

```
        ApplicationContainer apps = onoff.Install (star.GetSpokeNode(1));
        apps.Start (Seconds (1.0));
        apps.Stop (Seconds (9.0));
        //Uzel 3 - prijem
        PacketSinkHelper sink ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port)));
        apps = sink.Install (star.GetSpokeNode(3));
        apps.Start (Seconds (0.0));
```

Stejně jako v předchozí úloze, budeme i zde sledovat zpoždění jednotlivých datagramů, které si zobrazíme do grafu. Na každém rozhraní budeme také zachytávat provoz. Postupně tedy vkládejte známé příkazy. Také je opět nutné před hlavní funkci *main* zkopírovat obsah souboru *Database_star*.

```
        //Zachytavani .pcap
        pointToPoint.EnablePcapAll ("router");
```

```
        //Trasovani - aplikace
        Config::Connect("/NodeList/2/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "udp"));
        Config::Connect("/NodeList/4/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "udp"));
```

```
        //Volani saveLatency
        Simulator::Schedule(Seconds(10), &saveLatency);
```

Poslední vložený kód bude sloužit ke spuštění simulátoru.

```
        //Spusteni simulatoru
        Simulator::Stop (Seconds (10));
        Simulator::Run ();
        Simulator::Destroy ();
```

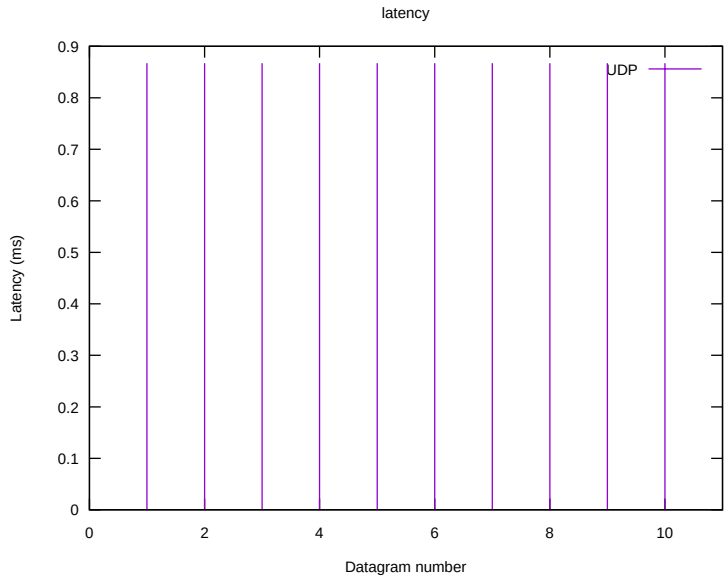
Na konci jsou opět umístěny řádky, které slouží ke zobrazení výstupu z terminálu a k vykreslení grafu. Ty je vhodné odkomentovat.

8.1.2.1 Spuštění simulátoru a zobrazení výsledků

Vytvořený kód znovu zkopírujeme do adresáře *scratch* a v terminálu simulaci spustíme známými příkazy. V případě správné funkčnosti se nám opět vypíší informace o zpoždění v síti. Výstup by měl vypadat takto.

```
UDP data sent: 5120 B
UDP data received: 5120 B, error rate: 0 %
Average latency for UDP: 0.8672 ms
Latency first datagram: 0.8672 ms
Latency fifth datagram: 0.8672 ms
```

V adresáři ns-3.27 se opět vykreslil graf zpoždění (Obr. 8-6).



Obr. 8-6: Zpoždění datagramu

Graf i hodnoty vypsané do terminálu si prohlédněte, odpovězte na následující otázky a splňte samostatné úkoly.

- Samostatně změňte počet terminálů připojených ke směrovači na 6 a 8. Jaký vliv to má na zpoždění?
- Jaký je rozdíl v případě komunikace v síti s přepínačem a v síti se směrovačem?

- Proč nedochází na začátku přenosu k vyššímu zpoždění, jako v případě přepínače?
- Jak proces směrování funguje? Co je třeba sestavit?
- Na jaké vrstvě směrovač pracuje a jaké adresy pro směrování používá?

8.1.2.2 Výpadek v síti

Velikost odesílaných dat aplikací bude neomezená. Aplikace tak bude omezena pouze časem. To uděláme editací proměnné *maxBytes*. Hodnotu nastavíme na 0. Zajímat nás teď budou pouze hodnoty přenesených dat a chybovost. Zakomentujte tak všechny položky v části //Vystup terminal, které souvisí se zpožděním. Stejně tak zakomentujte část //Zobrazeni grafu. A simulaci spustíme. Sledujte kolik dat je přeneseno a s jakou chybovostí.

```
UDP data sent: 999936 B
UDP data received: 999936 B, error rate: 0 %
Average latency for UDP: 0.8672 ms
Latency first datagram: 0.8672 ms
Latency fifth datagram: 0.8672 ms
```

Dále do simulace vložíme kód, který nám vytvoří výpadek rozhraní na centrálním uzlu v čase 5 sekund.

```
//Vypadek
Ptr<Ipv4> IPv4 = star.GetHub()->GetObject<Ipv4> ();
Simulator::Schedule(Seconds(5), &Ipv4::SetDown, IPv4, 2);
```

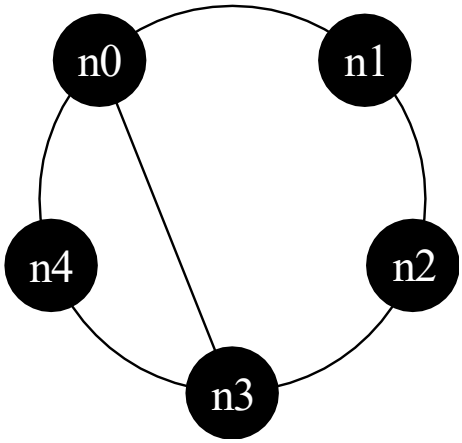
Simulátor můžeme opět spustit. Vystup z terminálu by měl vypadat takto. Odpovězte na doplňující otázky. **Kód simulace si opět uložte pro pozdější použití.**

```
UDP data sent: 999936 B
UDP data received: 499712 B, error rate: 50.0256 %
Average latency for UDP: 0.8672 ms
Latency first datagram: 0.8672 ms
Latency fifth datagram: 0.8672 ms
```

- Jak je topologie hvězda odolná k výpadkům centrálního uzlu?
- Co se stane v případě výpadku jednoho z ostatních uzlů?

8.2 Tvorba modelu – obecná topologie

V případě obecné topologie (vychází z kruhové topologie) nenabízí ns-3 žádné ulehčení tvorby jako v předchozí úloze (*PointToPointStarHelper*). Topologii, která je uvedena na obrázku, tedy vytvoříme ručně postupným zadáváním příkazů.



Obr. 8-7: Obecná topologie

Kód budeme opět vkládat do předpřipraveného souboru *ring_vychozi.cc*. Začneme tvorbou pěti uzlů.

```
//Vytvoreni uzlu
NodeContainer n;
n.Create (5);
```

Mezi uzly budeme používat jednoduchou linku bod-bod. Její definice je následovná.

```
//Definice linky
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mb/s"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("0ms"));
```

Dále definujeme vazby mezi uzly. **Vazbu mezi uzlem n3 a n0 vytvořte samostatně** *NodeContainer n3n0).*

```
//Vazba mezi uzly
NodeContainer n0n1 = NodeContainer (n.Get (0), n.Get (1));
NodeContainer n1n2 = NodeContainer (n.Get (1), n.Get (2));
NodeContainer n2n3 = NodeContainer (n.Get (2), n.Get (3));
```

```
NodeContainer n3n4 = NodeContainer (n.Get (3), n.Get (4));
NodeContainer n4n0 = NodeContainer (n.Get (4), n.Get (0));
```

Mezi rozhraní uzlů přiřadíme dříve definovanou *pointToPoint* linku. **Mezi uzly n3 a n0 ji opět přiřaďte samostatně** (*NetDeviceContainer d3d0*).

```
//Prirazeni linky mezi rozhrani
NetDeviceContainer d0d1 = pointToPoint.Install (n0n1);
NetDeviceContainer d1d2 = pointToPoint.Install (n1n2);
NetDeviceContainer d2d3 = pointToPoint.Install (n2n3);
NetDeviceContainer d3d4 = pointToPoint.Install (n3n4);
NetDeviceContainer d4d0 = pointToPoint.Install (n4n0);
```

Před přiřazením IP adres nainstalujeme protokolový zásobník na všechny uzly známým kódem.

```
//Internet Stack
InternetStackHelper internet;
internet.Install (n);
```

Samotné přiřazení IP adres probíhá následujícím kódem. Protože se jedná o spojení dvou bodů využíváme masku 255.255.255.252. **Opět samostatně přiřaďte vhodnou adresu na rozhraní mezi uzly n3 a n0.**

```
//Prirazeni IP adres
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.0.1.0", "255.255.255.252");
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);
ipv4.SetBase ("10.0.2.0", "255.255.255.252");
Ipv4InterfaceContainer i1i2 = ipv4.Assign (d1d2);
ipv4.SetBase ("10.0.3.0", "255.255.255.252");
Ipv4InterfaceContainer i2i3 = ipv4.Assign (d2d3);
ipv4.SetBase ("10.0.4.0", "255.255.255.252");
Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);
ipv4.SetBase ("10.0.5.0", "255.255.255.252");
Ipv4InterfaceContainer i4i0 = ipv4.Assign (d4d0);
```

Protože budeme chtít v síti komunikovat, povolíme směrování známým příkazem.

```
//Povoleni smerovani v siti
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Komunikace bude probíhat z uzlu n2 na uzel n1. Podle toho je vytvořena následující UDP aplikace. Ta nemá žádné omezení z hlediska velikosti přenesených dat. Omezuje ji tak pouze čas.

```
	//UDP Aplikace
	uint16_t port = 9;   // Discard port (RFC 863)
	//n2 - odesilani
	OnOffHelper onoff ("ns3::UdpSocketFactory",Address (InetSocketAddress
(i0i1.GetAddress (1), port)));
	onoff.SetConstantRate (DataRate ("1Mb/s"));
	ApplicationContainer apps = onoff.Install (n.Get (2));
	apps.Start (Seconds (1.0));
	apps.Stop (Seconds (20.0));
	//n1 - prijem
	PacketSinkHelper sink ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port)));
	apps = sink.Install (n.Get (1));
	apps.Start (Seconds (0.0));
```

Výsledky si opět zobrazíme jako výstup z terminálu, zachycené soubory *.pcap a* graf. Opět je tedy nutné zkopírovat před hlavní *main* obsah souboru *Database_topo* a doplnit simulaci o volání.

```
	//Zachytavani Pcap
	pointToPoint.EnablePcapAll ("ring");
	//Trasovi - aplikace
	Config::Connect("/NodeList/2/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "udp"));
	Config::Connect("/NodeList/1/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "udp"));
	//Volani saveLatency
	Simulator::Schedule(Seconds(20), &saveLatency);
```

Simulátor bude tentokrát spuštěn na 20 sekund, stejně jsou nastaveny i aplikace.

```
	//Spusteni simulatoru
	Simulator::Stop (Seconds (20));
	Simulator::Run ();
	Simulator::Destroy ();
```

Na konci souboru jsou opět zakomentované položky, sloužící k vykreslení grafu a vypsání výstupů do terminálu. Ty odkomentujte.

8.2.1 Spuštění simulátoru a zobrazení výsledků

Nyní můžete známými příkazy spustit simulátor. V terminálu by se měly vypsat následující informace.

```
UDP data sent: 2.37466e+06 B
UDP data received: 2.37466e+06 B, error rate: 0 %
Average latency for UDP: 0.4336 ms
```

Ze zachycených souborů zjistěte, na které lince komunikace probíhá. Poté můžete pokračovat na další kapitulu, ve které budeme nastavovat výpadek.

8.2.2 Výpadek v síti

V síti dojde, podle našeho nastavení, ke dvěma výpadkům. První nastane v čase 10 sekund a dojde k výpadku rozhraní číslo 1 na uzlu n2 (linka n3-n0). K tomu slouží následující kód.

```
//Vypadek
Ptr<Ipv4> IPv4 = n.Get(2)->GetObject<Ipv4> ();
Simulator::Schedule(Seconds(10), &Ipv4::SetDown, IPv4, 1);
```

Protože používáme globální směrování ns-3, které je bez signalizace, musíme po výpadku ručně přepočítat směrovací tabulky. To uděláme následujícím kódem, který vložíme hned za kód, kterým výpadek nastavujeme.

```
Simulator::Schedule (Seconds (10),&Ipv4GlobalRoutingHelper::RecomputeRoutingTables);
```

Druhý výpadek nastane na rozhraní číslo 2 uzlu n3 (linka n3-n0) v čase 15 sekund. K tomu vložíme následující příkazy (rozhrání jsou číslována v pořadí, v jakém jsou vytvořena, dejte tedy pozor, v jakém pořadí jste rozhraní na uzlu vytvářeli).

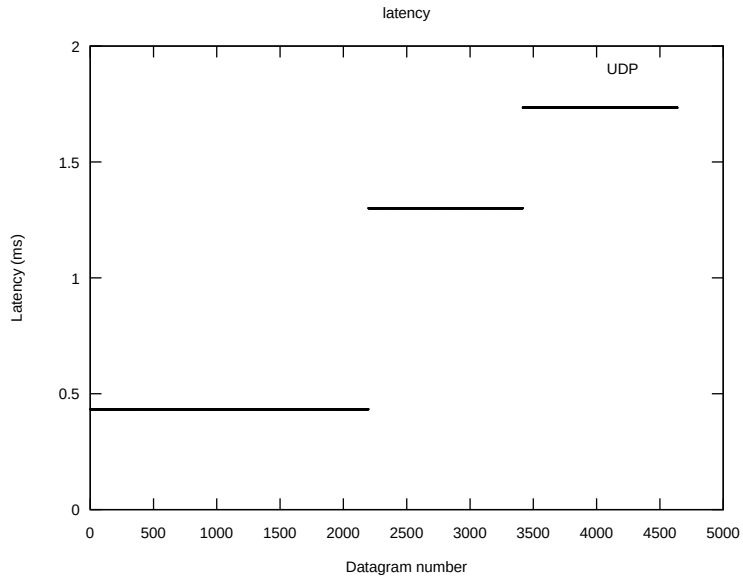
```
Ptr<Ipv4> ipV4 = n.Get(3)->GetObject<Ipv4> ();
Simulator::Schedule(Seconds(15), &Ipv4::SetDown, ipV4, 3);
```

V čase 15 sekund je opět nutné přepočítat směrovací tabulky. To udělejte samostatně.

Simulátor opět spusťte a na výstupu z terminálu si všimněte změny zpoždění.

```
UDP data sent: 2.37466e+06 B
UDP data received: 2.37466e+06 B, error rate: 0 %
Average latency for UDP: 1.00416 ms
```

V adresáři ns-3.27 se nám také vytvořil graf (Obr. 8-8). Graf popište a odpovězte na následující otázky.

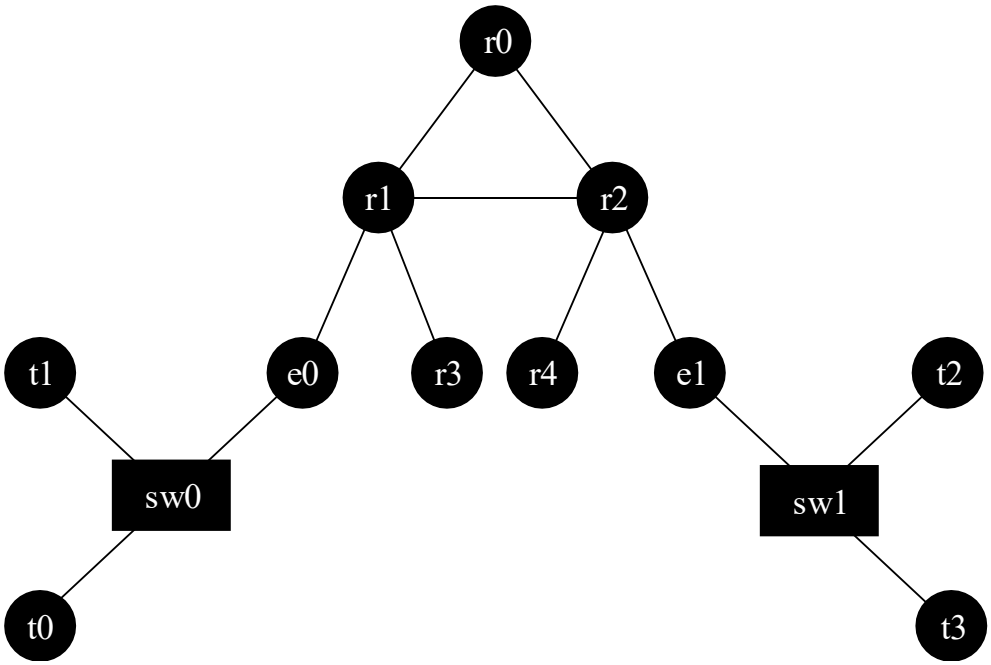


Obr. 8-8: Graf zpoždění při výpadku

- Proč se mění zpoždění?
- Kudy se data směrují v případě výpadku? Ověřte ze zachycených souborů *.pcap*.
- Jaká je odolnost obecné topologie proti výpadku? Srovnejte s hvězdou.

8.3 Tvorba modelu – strom

V poslední části laboratorní úlohy vytvoříme jednoduchou stromovou síť (Obr. 8-9). Síť bude obsahovat několik směrovačů, přepínačů a několik koncových terminálů. Síť podle návodu sestavíme a otestujeme použitím globálního směrování. V druhém kroku povolíme a nakonfigurujeme RIPv2.



Obr. 8-9: Stromová topologie

Proto je v této síti relativně velké množství uzlů, každý zvlášť si pojmenujeme abychom měli konfiguraci přehlednější. Z toho důvodu se ale bude kód mírně lišit od předchozích úloh. Otevřete si soubor *tree_vychozi.cc*. Tento soubor budeme dále rozšiřovat a upravovat. Při konfiguraci nám pomůže Tab. 8-1, ve které jsou vypsány jednotlivé linky, IP adresa a typ linky.

Tab. 8-1: Přiřazení adresních rozsahů k linkám

Linka	IP rozsah	Typ
r0r1	10.1.0.0/30	Point-to-point
r0r2	10.1.1.0/30	
r1r2	10.1.2.0/30	
r1r3	10.1.3.0/30	
r2r4	10.1.4.0/30	
e0r1	10.1.5.0/30	
e1r2	10.1.6.0/30	

sw0t0	10.0.10.0/24	csma
sw0t1		
sw0e0		
sw1e2	10.0.20.0/24	
sw1t3		
sw1e1		

Začneme vytvořením uzlů. Každý uzel vytvoříme zvlášť a pojmenujeme podle Obr. 8-9. To stejné platí i pro přepínače. Nejprve vytvoříme čtyři terminály a dva krajní směrovače, které budou později připojené k přepínači. Pokračovat budeme pěti směrovači a dvěma přepínači.

```
//Tvorba uzlu - terminal
Ptr<Node> t0 = CreateObject<Node> ();
Ptr<Node> t1 = CreateObject<Node> ();
Ptr<Node> t2 = CreateObject<Node> ();
Ptr<Node> t3 = CreateObject<Node> ();
```

```
//Tvorba uzlu - krajni smerovac
Ptr<Node> e0 = CreateObject<Node> ();
Ptr<Node> e1 = CreateObject<Node> ();
```

```
//Tvorba uzlu - smerovac
Ptr<Node> r0 = CreateObject<Node> ();
Ptr<Node> r1 = CreateObject<Node> ();
Ptr<Node> r2 = CreateObject<Node> ();
Ptr<Node> r3 = CreateObject<Node> ();
Ptr<Node> r4 = CreateObject<Node> ();
```

```
//Tvorba uzlu - prepinac
Ptr<Node> sw0 = CreateObject<Node> ();
Ptr<Node> sw1 = CreateObject<Node> ();
```

V úloze budeme používat dva typy linek *pointToPoint* pro spojení mezi směrovači a linku *csma* k propojení přepínačů s terminály. Obě linky si nyní nadefinujeme.

```
//Definice linky csma
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("10Mb/s"));
csma.SetChannelAttribute ("Delay", StringValue ("1ms"));
//Definice linky pointTopoint
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mb/s"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("1ms"));
```

Dále vytvoříme vazbu mezi uzly směrovačů. Názvy *NodeContainer* odpovídají uzlům, které spojují.

```
//Vazba mezi uzly
NodeContainer r0r1 = NodeContainer (r0, r1);
NodeContainer r0r2 = NodeContainer (r0, r2);
NodeContainer r1r2 = NodeContainer (r1, r2);
NodeContainer r1r3 = NodeContainer (r1, r3);
NodeContainer r2r4 = NodeContainer (r2, r4);
NodeContainer e0r1 = NodeContainer (e0, r1);
NodeContainer e1r2 = NodeContainer (e1, r2);
```

V dalším kroku nainstalujeme *pointToPoint* linky.

```
//Point-to-point
NetDeviceContainer dr0r1 = pointToPoint.Install (r0r1);
NetDeviceContainer dr0r2 = pointToPoint.Install (r0r2);
NetDeviceContainer dr1r2 = pointToPoint.Install (r1r2);
NetDeviceContainer dr1r3 = pointToPoint.Install (r1r3);
NetDeviceContainer dr2r4 = pointToPoint.Install (r2r4);
NetDeviceContainer de0r1 = pointToPoint.Install (e0r1);
NetDeviceContainer de1r2 = pointToPoint.Install (e1r2);
```

Nyní začneme konfigurovat přepínače. Následující kód znáte z kapitoly 8.1.1. Zde již bez vysvětlení. Pro sw0 je kód následující.

```
//Prepinac leva
NetDeviceContainer leftLanDevices;
NetDeviceContainer leftSwitchDevices;
NodeContainer leftLan (e0, t0, t1);
for (int i = 0; i < 3; i++){
 NetDeviceContainer link = csma.Install (NodeContainer (leftLan.Get (i), sw0));
 leftLanDevices.Add (link.Get (0));
 leftSwitchDevices.Add (link.Get (1));}
```

Pro sw1:

```
//Prepinac prava
NetDeviceContainer rightLanDevices;
NetDeviceContainer rightSwitchDevices;
NodeContainer rightLan (e1, t2, t3);
for (int i = 0; i < 3; i++){
 NetDeviceContainer link = csma.Install(NodeContainer(rightLan.Get (i), sw1));
 rightLanDevices.Add (link.Get (0));
 rightSwitchDevices.Add (link.Get (1));}
```

A samotné přiřazení přepínací funkce na sw0 i sw1.

```
      //Prepinac
      BridgeHelper swtch;
      swtch.Install (sw0, leftSwitchDevices);
      swtch.Install (sw1, rightSwitchDevices);
```

Tyto řádky nám později pomůžou ke konfiguraci protokolu RIP, pro funkční globální směrování jsou zbytečně složité, nicméně plně funkční. Vytvoříme si několik kontejnerů, které budou sdružovat uzly směrovačů (*routerNodes*), krajní směrovače t2 a t3 (*edgeNodes)* a koncové terminály (*terminalNodes*).

```
      //Node container
      NodeContainer routerNodes (r0, r1, r2, r3, r4);
      NodeContainer edgeNodes (e0, e1);
      NodeContainer terminalNodes (t0, t1, t2, t3);
```

Protokolový zásobník nainstalujeme zvlášť pro směrovače a zvlášť pro koncové terminály takto.

```
//Protokolovy zasobnik - router
InternetStackHelper internet;
internet.Install (routerNodes);
internet.Install (edgeNodes);
//Protokolovy zasobnik - terminal
InternetStackHelper internetTerminal;
internetTerminal.Install (terminalNodes);
```

Nyní můžeme přiřadit adresní rozsahy. Pro lepší orientaci použijte Tab. 8-1.

```
//Prirazeni adresnich rozsahu
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.0.0", "255.255.255.252");
      ipv4.Assign (dr0r1);
ipv4.SetBase ("10.1.1.0", "255.255.255.252");
      ipv4.Assign (dr0r2);
ipv4.SetBase ("10.1.2.0", "255.255.255.252");
      ipv4.Assign (dr1r2);
ipv4.SetBase ("10.1.3.0", "255.255.255.252");
      ipv4.Assign (dr1r3);
ipv4.SetBase ("10.1.4.0", "255.255.255.252");
      ipv4.Assign (dr2r4);
ipv4.SetBase ("10.1.5.0", "255.255.255.252");
      ipv4.Assign (de0r1);
ipv4.SetBase ("10.1.6.0", "255.255.255.252");
      ipv4.Assign (de1r2);
ipv4.SetBase ("10.0.10.0", "255.255.255.0");
      ipv4.Assign (leftLanDevices);
ipv4.SetBase ("10.0.20.0", "255.255.255.0");
      ipv4.Assign (rightLanDevices);
```

Zbývá povolit směrování známým příkazem.

```
//Povoleni smerovani
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Pro kontrolu, jestli vše funguje správně, vytvoříme aplikaci, která bude zasílat data z jedné strany sítě na druhou (z t0 na t5). Opět použijeme UDP aplikaci. Její definice vypadá stejně jako v předchozích případech.

```
      //UDP Aplikace
      uint16_t port = 9;   // Discard port (RFC 863)
      //t0 - odesilani
      OnOffHelper onoff ("ns3::UdpSocketFactory",Address (InetSocketAddress
("10.0.20.3", port)));
      onoff.SetConstantRate (DataRate ("1Mb/s"));
      ApplicationContainer apps = onoff.Install (t0);
      apps.Start (Seconds (11.0));
      apps.Stop (Seconds (30.0));
      //t5 prijem
      PacketSinkHelper sink ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port)));
      apps = sink.Install (t3);
      apps.Start (Seconds (10.0));
```

Výstupem ze simulace budou znovu zachycené soubory *.pcap*. Povolíme trasování aplikace tak, aby bylo možné sledovat přenesená data a zpoždění. Opět použijeme známé příkazy. V úloze používáme dva typy linek, je tedy nutné zachytávat na každé z nich.

```
      //Zachytavani .pcap
      pointToPoint.EnablePcapAll ("rip-p2p");
      csma.EnablePcapAll ("rip-csma");
```

```
 //Trasovani - aplikace
Config::Connect("/NodeList/0/ApplicationList/*/$ns3::OnOffApplication/Tx",
MakeBoundCallback(&sendPacket, "udp"));
Config::Connect("/NodeList/3/ApplicationList/*/$ns3::PacketSink/Rx",
MakeBoundCallback(&receivePacket, "udp"));
```

Posledním krokem, který je třeba udělat, je vytvoření kódu, který spustí simulátor. Simulace bude tentokrát trvat 50 sekund.

```
//Spusteni simulatoru
Simulator::Stop (Seconds (90));
Simulator::Run ();
Simulator::Destroy ();
```

Před hlavní funkci *main* zkopírujte metody a databáze, které jsou umístěny v souboru *Databaze_tree*. Na konci kódu si odkomentujte část //Vystup terminal. Graf budeme vykreslovat později.

8.3.1 Spuštění simulace a zobrazení výsledků

Při vytváření kódu simulace jsme povolili globální směrování. S jeho pomocí si vyzkoušíme pouze funkčnost zapojení. Hotový kód zkopírujte do známého adresáře a známými příkazy v terminálu spusťte. Na výstupu byste měli vidět následující parametry.

```
UDP data sent: 2.37466e+06 B
UDP data received: 2.37363e+06 B, error rate: 0.043122 %
Average latency for UDP: 5.34013 ms
```

Ze zachycených souborů zjistěte, kudy provoz v síti prochází. Tímto je funkce ověřena a v síti nyní povolíme směrování pomocí protokolu RIPv2.

8.3.2 Konfigurace směrování RIPv2

Směrovací protokol RIPv2 je v ns-3 podporován včetně signalizace, na rozdíl od globálního směrovaní, které vychází z OSPF. Ukážeme si, jak rychle protokol vytvoří směrovací tabulky. Jakou používá metriku a jak reaguje na výpadek.

Nejprve pomocí třídy *RipHelper* povolíme směrování a v rámci toho odebereme krajní rozhraní na uzlech, které vedou do koncových sítí. Díky tomu nám do nich protokol RIP nebude zasahovat. To uděláme vložením následujícího kódu.

```
//RIPv2
RipHelper ripRouting;
```

```
        //RIPv2 - odebrana rozhrani
        ripRouting.ExcludeInterface (e0, 2);
        ripRouting.ExcludeInterface (e1, 2);

        Ipv4ListRoutingHelper listRH;
        listRH.Add (ripRouting, 0);
```

Vytvořený *listRH* přiřadíme protokolovému zásobníku na směrovačích. Přiřazení tedy bude vypadat následovně (červené položky už byly vloženy dříve).

```
        //Protokolovy zasobnik - router
        InternetStackHelper internet;
        internet.SetRoutingHelper (listRH);
        internet.Install (routerNodes);
        internet.Install (edgeNodes);
```

Protože nám RIP nebude směrovat v koncových sítích, je nutné na terminálech t0, t1. t4 a t5 nakonfigurovat výchozí bránu, což bude terminál t2 nebo t3. Následující kód popisuje, jak výchozí bránu nastavit na terminálu t0 a t1. Samostatně doplňte pro druhou koncovou síť.

```
//RIPv2 vychozi brana
Ptr<Ipv4StaticRouting> staticRouting;
//leftLan
staticRouting = Ipv4RoutingHelper::GetRouting <Ipv4StaticRouting> (t0->GetObject<Ipv4>
()->GetRoutingProtocol ());
staticRouting->SetDefaultRoute ("10.0.10.1", 1 );
staticRouting = Ipv4RoutingHelper::GetRouting <Ipv4StaticRouting> (t1->GetObject<Ipv4>
()->GetRoutingProtocol ());
staticRouting->SetDefaultRoute ("10.0.10.1", 1 );
```

V tomto kroku také odstraňte původní řádek, který povoloval globální směrování. Je označen popiskem //Povoleni smerovani. Prozatím také zakomentujte UDP aplikaci.

Kromě zachytávání souborů a trasování aplikací se budeme chtít podívat na směrovací tabulky a zatížení linky protokolem RIP. Směrovací tabulku si vypíšeme na krajním směrovači t2 každých deset sekund. Použijeme k tomu následující příkazy.

```
   //RIPv2 - smerovaci tabulky
   Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>
("tree-rip-routes.log", std::ios::out);
   ripRouting.PrintRoutingTableEvery (Seconds (3), e0,routingStream);
```

Směrovací tabulka se uloží do stejného adresáře, jako zachycené soubory, pod názvem *tree-rip-route.log*.

Zatížení si ukážeme na lince, na které později nebude probíhat provoz. Vybereme tedy jednu z linek mezi r2 – r3 nebo r2 – r4. Graf budeme vykreslovat z příchozího provozu. K trasovacímu systému na uzlu r3 se tedy připojíme takto.

```
//Trasovani zatizeni
Config::Connect("/NodeList/9/$ns3::Ipv4L3Protocol/Rx",MakeCallback (&trasmitPacket));
```

Uzly jsou číslovány v pořadí, v jakém jsou vytvořeny, proto je uzel r3 9. v pořadí. Ve funkci *trasmitPacket* je podmínka, kvůli které jsou uloženy pouze pakety, které nesou informace o sítích, nikoliv požadavky.

Nyní stačí jen zavolat funkci *saveThroughput* a odkometovat kód na konci, který slouží k vykreslení grafu.

```
      //Volani funkce pro mereni zatizeni
      Simulator::Schedule(Seconds(0), &saveThroughput);
```

8.3.2.1 Spuštění simulátoru a zobrazení výsledků

Simulátor opět spustíme a pokud vše proběhlo v pořádku zobrazíme si směrovací tabulky uzlu t2, které vypadají jako na Obr. 8-10.

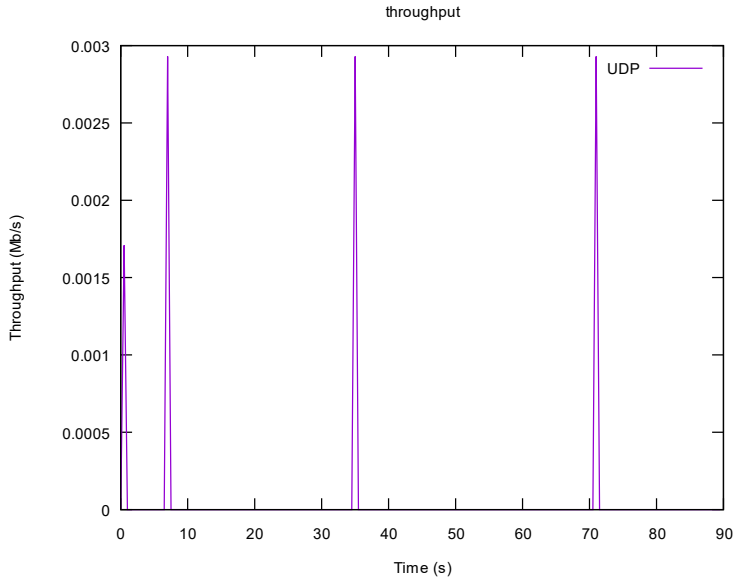
```
Node: 2, Time: +6.0s, Local time: +6.0s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 2, Time: +6.0s, Local time: +6.0s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.5.0        0.0.0.0         255.255.255.252 U     1      -      -   1
10.0.10.0       0.0.0.0         255.255.255.0   U     1      -      -   2

Node: 2, Time: +9.0s, Local time: +9.0s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 2, Time: +9.0s, Local time: +9.0s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.1.2.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.1.0.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.0.20.0       10.1.5.2        255.255.255.0   UGS   4      -      -   1
10.1.1.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
10.1.4.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
10.1.6.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
```

Obr. 8-10: Směrovací tabulka uzlu t2

Dále si otevřete graf zatížení linky, který je také na Obr. 8-11. Odpovězte na otázky a splňte úkoly.

- V jakém čase byla sestavena kompletní směrovací tabulka?
- Srovnejte uvedenou metriku s topologií.
- Otevřete si zachycený soubor na uzlu r3 (9. v pořadí) a srovnejte příchozí provoz s grafem.
- Jak často jsou zprávy přenášeny?
- Co je obsahem těchto zpráv?
- **Zprovozněte UDP aplikaci (z t0 na t5). Čas spuštění nastavte až po sestavení směrovacích tabulek (zhruba 10 sekund).**
    - Ze zachycených souborů zjistěte, kudy jsou datagramy směrovány.



Obr. 8-11: Přenos z uzlu r1 na uzel r3

8.3.3 Výpadek v síti

Nyní nastavíme výpadek v síti mezi uzly r1 a r2. Budeme sledovat, jak na tento výpadek protokol zareaguje, a zda bude směrování fungovat.

Před hlavní funkci *main* vložíme funkci *TearDownLink*, kterou budeme později volat.

```
void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB, uint32_t interfaceA,
uint32_t interfaceB)
       {
         nodeA->GetObject<Ipv4> ()->SetDown (interfaceA);
         nodeB->GetObject<Ipv4> ()->SetDown (interfaceB);
       }
```

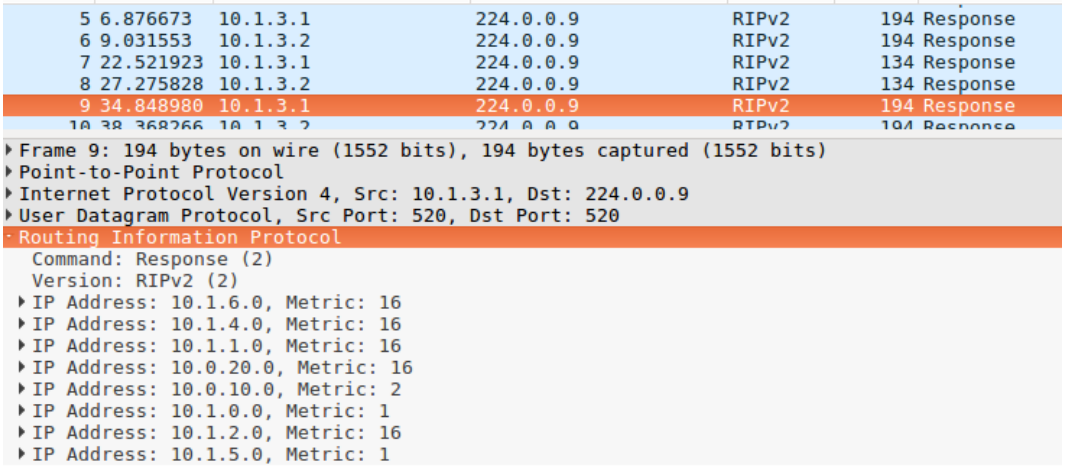
Výpadek budeme nastavovat v čase 20 sekund a funkci *TearDownLink* předáme dva uzly a rozhraní ke kterým linka náleží. V našem případě bude tedy kód vypadat takto.

```
      //Volani funkce pro vypadek
      Simulator::Schedule (Seconds (20), &TearDownLink, r1, r2, 2, 2);
```

Vypnutí aplikace nastavte na 80 sekund a spuštění nechejte tak, jak jste jej nastavili v předchozím úkolu.

8.3.3.1 Spuštění simulátoru a zobrazení výsledků

Simulátor znovu spusťte a v zachycených souborech nalezněte, jak se směrovače informují o nedostupnosti (Obr. 8-12). Také si otevřete směrovací tabulku (Obr. 8-13). Odpovězte na otázky a splňte úkoly.



Obr. 8-12: Provoz zachycený na uzlu r3

```
  Priority: 0 Protocol: ns3::Rip
Node: 2, Time: +18.0s, Local time: +18.0s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.1.2.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.1.0.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.0.20.0       10.1.5.2        255.255.255.0   UGS   4      -      -   1
10.1.1.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
10.1.4.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
10.1.6.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
10.1.5.0        0.0.0.0         255.255.255.252 U     1      -      -   1
10.0.10.0       0.0.0.0         255.255.255.0   U     1      -      -   2

  Priority: 0 Protocol: ns3::Rip
Node: 2, Time: +48.0s, Local time: +48.0s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.3.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.1.0.0        10.1.5.2        255.255.255.252 UGS   2      -      -   1
10.0.20.0       10.1.5.2        255.255.255.0   UGS   5      -      -   1
10.1.1.0        10.1.5.2        255.255.255.252 UGS   3      -      -   1
10.1.4.0        10.1.5.2        255.255.255.252 UGS   4      -      -   1
10.1.6.0        10.1.5.2        255.255.255.252 UGS   4      -      -   1
10.1.5.0        0.0.0.0         255.255.255.252 U     1      -      -   1
10.0.10.0       0.0.0.0         255.255.255.0   U     1      -      -   2
```

Obr. 8-13: Směrovací tabulka uzlu t2 před výpadkem a po reakci na něj

- Kudy byl směrován provoz po výpadku a jak se to projevilo na metrice?
- Cestu datagramů ověřte ve Wiresharku.
- Jaké zprávy byly přenášeny po výpadku, co obsahují?
- Jaký je význam metriky 16?
- Jak zjistí terminály, kam datagramy zasílat, když na nich není zapnuto směrování? Tyto informace nalezněte v zachycených souborech a vysvětlete.
- Zjistěte informace o směrovacím protokolu OSPF a srovnejte ho s RIP.

9. ZÁVĚR

Cílem diplomové práce bylo kompletně zpracovat tři laboratorní úlohy pro výuku síťových technologií a protokolů. Jako simulační prostředí byl zvolen ns-3 s doplňujícími nástroji DCE a Quagga. Úlohy se zabývají základní konfigurací protokolu BGP, transportními protokoly TCP, UDP, SCTP a síťovými prvky, topologiemi a směrováním pomocí RIPv2.

První část práce je zaměřená na popis simulátoru ns-3 a jsou zde rozebrány základní objekty používané simulátorem. Popsána je také metoda přímého vykonávání kódu, kterou v ns-3 zajišťuje nástroj DCE. Podporu samotného směrovacího protokolu BGP zajišťuje balík Quagga.

Každá laboratorní úloha se vždy skládá z teoretického úvodu a samotného návodu k vypracování simulace v ns-3, včetně doplňujících otázek a samostatných úkolů.

Výstupy první simulace, která se zabývá směrovacím protokolem BGP, jsou směrovací tabulky, grafy zatížení linky a soubory zachycené na jednotlivých rozhraních. BGP je v ns-3 podporováno jen velmi omezeně. Není možné využívat jinou metriku než počet procházených autonomních systémů. Nejsou podporovány ani další parametry cesty, jako například lokální preference nebo výstupní diskriminátor. Zatím také není implementován proces redistribuce směrovacích informací mezi externím a interním směrovacím systémem.

Druhá úloha je zaměřená na srovnání transportních protokolů TCP a UDP, jejich reakce na chybovost, výpadky a přetížení v síti. Výstupem jsou grafy zatížení uzlů na vysílací i přijímací straně, a také soubory zachycené na rozhraních. Jsou také doplněny funkce pro měření zpoždění a dalších podobných parametrů.

Poslední úloha se nejdříve zabývá srovnáním směrovače a přepínače. Simulované jsou některé druhy topologií a předveden je také protokol ARP. V poslední části je vytvořena stromová struktura sítě, která obsahuje směrovače i přepínače. Na této síti je následně povoleno směrování směrovacím protokolem RIPv2. Kromě zachycených souborů jsou výstupem směrovací tabulky a grafy zatížení.

10. LITERATURA

[1] Nsnam. *Ns-3 network simulator* [online]. b.r. [cit. 2018-12-03]. Dostupné z: https://www.nsnam.org/about/

[2] *Ns-3 Tutorial* [online]. Release ns-3.29. 2018 [cit. 2018-12-03]. Dostupné z: https://www.nsnam.org/docs/release/3.29/tutorial/ns-3-tutorial.pdf

[3] *Ns-3 manual* [online]. 2018 [cit. 2018-12-04]. Dostupné z: https://www.nsnam.org/docs/manual/ns-3-manual.pdf

[4] *Conceptual Overview: ns-3* [online]. b.r. [cit. 2018-12-05]. Dostupné z: https://www.nsnam.org/docs/tutorial/html/conceptual-overview.html

[5] *Ns-3 Direct Code Execution (DCE): manual* [online]. b.r. [cit. 2018-12-05]. Dostupné z: https://www.nsnam.org/docs/dce/manual/ns-3-dce-manual.pdf

[6] Quagga Routing Suite. *Quagga* [online]. b.r. [cit. 2018-12-05]. Dostupné z: https://www.quagga.net/

[7] *Ns-3 Direct Code Execution (DCE) Quagga Manual: Direct Code Execution project* [online]. Release 1.7. 2016 [cit. 2018-12-01].

[8] *Ns-3 Direct Code Execution (DCE) Manual* [online]. 2016 [cit. 2019-05-14]. Dostupné z: ns-3-dce-manual.pdf

[9] *Using your in-kernel protocol implementation* [online]. b.r. [cit. 2019-03-14]. Dostupné z: https://ns-3-dce.readthedocs.io/en/latest/dce-user-kernel.html

[10] GRYGAREK, Petr. *Protokoly třídy Distance Vector* [online]. b.r. [cit. 2018-12-01]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/projekty0405/RouteOptimization/dokumentace/ar01s01.html

[11] GRYGAREK, Petr. *Protokol OSPF* [online]. b.r. [cit. 2018-12-01]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/lect/OSPF/ospf.html

[12] JEŘÁBEK, Jan. *Pokročilé komunikační techniky* [online]. Brno: Vysoké učení technické v Brně, 2015 [cit. 2018-12-01].

[13] BEIJNUM, Iljitsch van. *BGP*. 1st edition. Sebastopol, CA: O'Reilly, 2002. ISBN 05-960-0254-8.

[14] ZHANG, Randy a Micah BARTELL. *BGP design and implementation*. First Printing. Indianapolis, IN: Cisco Press, 2004. Cisco Press networking technology series. ISBN 15-870-5109-5.

[15] REKHTER, Yakov, T. LI a S. HARES. *A Border Gateway Protocol 4 (BGP-4): RFC 4271* [online]. 2006 [cit. 2018-12-01]. Dostupné z: https://tools.ietf.org/html/rfc4271

[16] FOROUZAN, Behrouz A. *TCP/IP protocol suite*. 4th ed. Boston: McGraw-Hill Higher Education, 2010. ISBN 978-0-07-337604-2.

[17] JEŘÁBEK, Jan. *Komunikační technologie* [online]. Brno, 2013 [cit. 2019-05-01].

[18] POSTEL, J. *RFC 768: User Datagram Protocol* [online]. 1980 [cit. 2019-04-19]. Dostupné z: https://tools.ietf.org/html/rfc768

[19] *RFC 793: TRANSMISSION CONTROL PROTOCOL* [online]. 1981 [cit. 2019-03-13]. Dostupné z: https://tools.ietf.org/html/rfc793#section-3.1

[20] STEWART, R. *Stream Control Transmission Protocol* [online]. 2007 [cit. 2019-05-02]. Dostupné z: https://tools.ietf.org/html/rfc4960

[21] MALKIN, G. *RFC 2453: RIP Version 2* [online]. 1998 [cit. 2019-05-05]. Dostupné z: https://tools.ietf.org/html/rfc2453

Obsah přiloženého CD

- Výchozí soubory:
  - bgp_basic.cc
  - bgp_basic_sam.cc
  - TCP_UDP_vychozi.cc
  - DatabaseTCP_UDP.txt
  - switch_vychozi.cc
  - router_vychozi.cc
  - ring_vychozi.cc
  - Database_topo.txt
  - tree_vychozi.cc
  - Database_tree.txt
- Návody k úlohám
  - uloha1_bgp.docx
  - uloha2_transportni_protokoly.docx
  - uloha3_sitove-prvky_topologie_rip.docx